Innehåll

1 Tillämpade numeriska metoder

1.1 Inledning

Numeriska beräkningar kommer in i alla tekniska tillämpningar och det gäller att angripa problemen effektivt och med tillförlitliga metoder. De numeriska basmetoderna har presenterats i grundkursen i numeriska metoder. I en fortsättningskurs är det naturligt att införa flera numeriska verktyg för att få möjlighet till numerisk behandling av lite mer avancerade ingenjörsmässiga tillämpningsproblem.

Avsikten med detta kompendium är att nödtorftigt täcka en fyrapoängs fortsättningskurs i numeriska metoder. För några kursavsnitt hänvisas till *Gerd Eriksson, Numeriska algoritmer med Matlab (NAM).*

Vissa moment såsom linjär och ickelinjär modellanpassning innebär nyttig repetition av grundkursstoffet. Det kan i viss mån också gälla avsnittet om formgivning av kurvor och rymdytor med hjälp av parametriska kurvor (splines och bézierkurvor).

Numerisk linjär algebra är ett viktigt moment. Hit hör goda algoritmer för lösning av stora glesa ekvationssystem och algoritmer för egenvärden till matriser, liksom QR-faktorisering och singulärvärdesfaktorisering. Den sistnämnda matrisfaktoriseringen är ett användbart numeriskt redskap vid bland annat datortomografi, bildkomprimering och beräkning av robotrörelser.

Numerisk behandling av ordinära differentialekvationer (ODE) är centralt i kursen; allt från begynnelsevärdesproblem med explicita och implicita stegmetoder till randvärdes- och egenvärdesproblem med finitadifferensmetoder och ansatsmetoder. Därefter är steget inte långt till de partiella differentialekvationerna (PDE) som omfattar potentialproblem, vågutbredningsproblem och värmeledningsproblem.

Frekvensanalys av samplade periodiska data med diskret fouriertransform (DFT) behandlas och orientering om snabb fouriertransform (FFT) ges.

Kompletterande kursbok – nödvändig för vissa avsnitt – är *Cleve B Moler, Numerical Computing with Matlab* (2004).
En bok som fördjupar teorin där kompendiet är ytligt är *Michael T Heath, Scientific Computing – An Introductory Survey* (2002).

På 2D1220-kursens hemsida

`http://www.nada.kth.se/kurser/kth/2D1220/`

finns information om kursen och ytterligare förslag på fördjupnings- och referenslitteratur.

1.2 Linjära och ickelinjära ekvationssystem

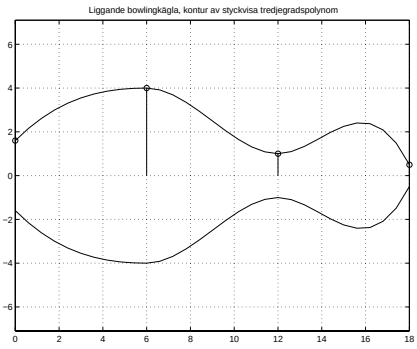
Linjära ekvationssystem $\mathbf{Ax} = \mathbf{b}$ behandlas i NAM avsnitt 1.4 – 1.8. Gausselimination och dess effektivitet för lösning av system med olika slag av systemmatriser studeras där. Iterativa metoder för system med diagonaltung matris behandlas i NAM avsnitt 6.7.

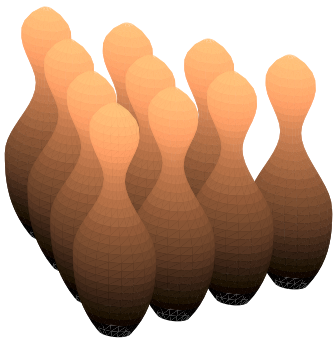
Några metoder för lösning av ickelinjära ekvationssystem beskrivs i NAM avsnitt 6.8 – 6.9; allra viktigast är Newtons metod. Fräscha upp kunskaperna genom att studera dessa avsnitt.

1.3 Hermiteinterpolation, kubiska splines och fusksplines

Givet ett antal punkter $(x_i, y_i),\ i = 1, 2, ..., n,$ sök en mjukt interpolerande kurva genom punkterna. Enklaste sättet att åstadkomma en kontinuerlig kurva är att utföra styckvis interpolation med förstagradspolynom, men det ger en kantig kurva med diskontinuitet i derivatan vid varje interpolationspunkt.

Styckvis interpolation med tredjegradspolynom ger mycket mjukare kurvkontur. Med hjälp av hermiteinterpolation kan vi lägga kurvor såsom fusksplines eller äkta kubiska splines genom punkterna. Studera detta i NAM avsnitt 3.3.



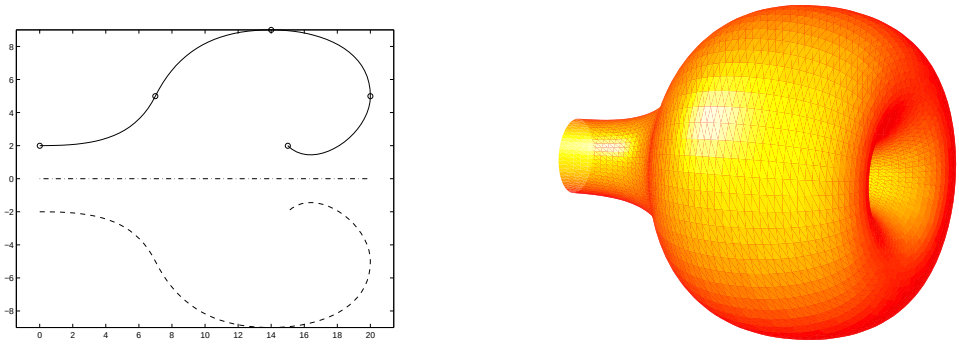


Figurens bowlingkäglor erhålls genom att konturen som skapats av styckvisa tredjegradspolynom (med hermiteinterpolation) roteras kring x-axeln. I MATLAB är det sedan lätt att resa dem upp (byta x och z), och rita tio stycken utplacerade i en triangel.

Interpolerande rymdytor över ett givet gitter av x- och y-värden kan man skapa med hjälp av bikubisk interpolation, se NAM avsnitt 3.7.

1.4 Parametriska kurvor — bézierkurvor och B-splines

Kapitel 4 i NAM ägnas åt bézierkurvor, en lätthanterlig klass av parametriska polynom med trevliga geometriska egenskaper. Kvadratiska och kubiska bézierkurvor har stor användning vid geometrisk modellering.



Den heldragna kurvan i vänstra bilden är skapad av fyra kubiska bézierkurvor mellan fem givna punkter (med givna kurvlutningar). Den undre kurvan är speglingen i x-axeln. När kurvan roteras kring x-axeln får vi den högra skapelsen.

I NAM avsnitt 4.4 går man vidare från interpolerande styckvisa bézierkurvor till ickeinterpolerande B-splines, som har andraderivatakontinuitet överallt. Studera avsnittet för att få en orientering om B-splines.

2 Optimering

2.1 Maximering/minimering i envariabelfallet

Om funktionen $F(x)$ som vi söker maximum eller minimum för är deriverbar två gånger, får vi x-värdet för extrempunkten genom att lösa ekvationen $F'(x) = 0$ med Newton-Raphsons metod med lämpligt val av startgissning.

För en funktion som är unimodal i intervallet $a \leq x \leq b$ men som är omöjlig (eller näst intill omöjlig) att derivera finns den derivatafria algoritmen *gyllenesnittetsökning* för beräkning av maximum eller minimum. Metoden beskrivs i NAM avsnitt 7.1. Algoritmen för minimering av $F(x)$ lyder:

- Bilda gyllenesnittetstorheten $r_g = (\sqrt{5} - 1)/2$ samt $q_g = 1 - r_g$
- Utgå från ett intervall $a \leq x \leq b$ 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.

I varje iteration i gyllenesnittetsökningen krymper intervallet med faktorn r_g som enligt ovan är $\frac{1}{2}(\sqrt{5}-1) = 0.618$.

För att kunna använda algoritmen för maximering i stället behövs en liten men väsentlig ändring: byt villkoret $F_1 < F_2$ mot $F_1 > F_2$.

2.2 Maximering/minimering i flervariabelfallet

Vid maximering eller minimering av flervariabelfunktionen $Q(x_1, x_2, ..., x_n)$ måste följande samband gälla vid extrempunkten:

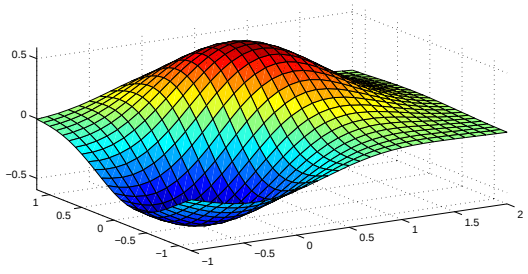
$$\partial Q/\partial x_1 = 0,\ \ \partial Q/\partial x_2 = 0,\ \ldots,\ \partial Q/\partial x_n = 0,$$

det vill säga gradienten ∇Q måste vara lika med noll vid den sökta punkten. Valet av lösningsalgoritm beror på hur pass enkelt eller omöjligt det är att få fram andraderivatorna till funktionen Q.

2.2.1 Newtons metod på gradientekvationen

De partiella derivatorna $\partial Q/\partial x_i,\ i = 1, ..., n$, satta till noll utgör tillsammans ett ickelinjärt ekvationssystem $\nabla Q(x_1, x_2, ..., x_n) = 0$. För att lösa ett sådant system kan man tillämpa Newtons metod, men det kräver att jacobianmatrisen **J** kan ställas upp. Matriselementen utgörs av andraderivatorna, $J_{ik} = \partial^2 Q/\partial x_i \partial x_k$ (denna matris kallas även hessianmatris).

Om andraderivatorna existerar och är enkla att beräkna är Newtons metod ypperlig för optimeringsproblemet.
I NAM avsnitt 7.2 studeras rymdytan $Q(x, y)$ i figuren, $Q(x, y) = (x + \sin y)e^{-x^2-y^2}$ och maxpunkten bestäms med Newtons metod. Studera detta!



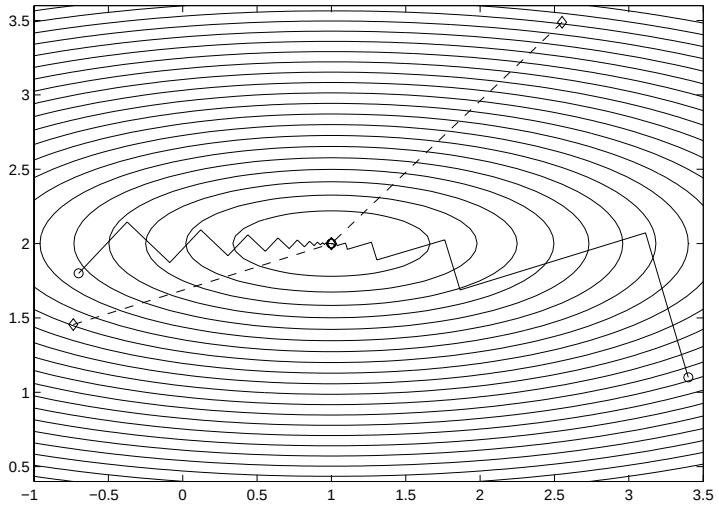
2.2.2 Steepestdescentmetoden

Steepestdescentmetoden för minimering föreslogs av matematikern A.Cauchy redan 1845. Algoritmen kan generaliseras och tillämpas på optimeringsproblem i många variabler, men det är för funktioner i två variabler som vi kan få en god föreställning om hur den arbetar.

Betrakta grafen över funktionen $Q(x, y)$ som ett landskap med kullar och dalar. Längst ner i varje dal har funktionen ett lokalt minimum. Om vi inte redan befinner oss i ett minimum men vill hamna där, ska vi naturligtvis välja en nedåtgående stig.

När man befinner sig i punkten (x_0, y_0) anger gradientvektorn $\nabla Q(x_0, y_0)$ den riktning som har brantast lutning. Det gäller alltså att gå i negativa gradientriktningen för att få största möjliga nedförsbacke ("steepest descent") lokalt sett. I den algoritm som kallas för steepestdescentmetoden traskar man på i oförändrad riktning, $-\nabla Q(x_0, y_0)$, så länge det bär nedåt. I detta läge, (x_1, y_1), har vi nått minimum längs denna bana.

Nu är det dags att göra en ny gradientberäkning $\nabla Q(x_1, y_1)$ och traska vidare i negativ gradientriktning tills det inte sluttar nedåt längre. Detta upprepas tills gradientvektornormen är tillräckligt nära noll.



Bilden visar en uppsättning nivåkurvor för det skålformade landskapet $Q(x, y) = (x-1)^2 + 9(y-2)^2 + 1$. Längs varje nivåkurva gäller att $Q(x, y)$ är konstant; vi får en kartbild över området. Två trappstegsformade steepestdescentpromenader är inritade, den vänstra startar i $x = -0.7$, $y = 1.8$, och den högra har startpunkten $x = 3.4$, $y = 1.1$.
(De streckade linjerna är lösning med Newtons metod, mer om detta senare.)

I varje steg i steepestdescentmetoden gäller det att finna minimum längs en sökriktning. Det är ett endimensionellt minimimeringsproblem som benämns *linjesökning*. Från punkten $\mathbf{p}_k = (x_k, y_k)$ i iterationssteg k ska vi förflytta oss i riktningen $\mathbf{g}_k = -\nabla Q(x_k, y_k)$ längs en linje som i vektorform lyder $\mathbf{p}_k + s\,\mathbf{g}_k$. Om komponenterna i $\mathbf{g}_k$ betecknas med g_x och g_y kan vi skriva linjen i parameterform: $x(s) = x_k + s\,g_x$ och $y(s) = y_k + s\,g_y$, där s alltså är den enda storheten som tillåts variera, och det gäller att $s \geq 0$. (Varför det?)

Linjesökningen består alltså i att finna minimum av envariabelfunktionen $F(s) = Q(x_k + sg_x,\, y_k + sg_y)$ i ett intervall $0 \leq s \leq s_{max}$.

Om $F(s)$ är svår eller kanske omöjlig att derivera analytiskt är gyllenesnittetsökning lämplig för linjesökningen. Om däremot $F(s)$ är lätt att derivera, så beräknar vi s-värdet och tillhörande minimum genom att lösa ekvationen $F'(s) = 0$.

Exempel: I minimeringsproblemet för $Q(x, y) = (x-1)^2 + 9(y-2)^2 + 1$ bestäms $F(s)$ i iterationssteg k av uttrycket

$$F(s) = (x_k + s\,g_x - 1)^2 + 9(y_k + s\,g_y - 2)^2 + 1,$$

där $g_x = -2(x_k - 1)$ och $g_y = -18(y_k - 2)$ (negativa gradientvektorns komponenter). Derivering ger $F'(s) = 2(x_k + s\,g_x - 1)g_x + 18(y_k + s\,g_y - 2)g_y$, som är ett förstagradspolynom i s. Derivatan blir noll då

$$s = s_k = -\frac{(x_k - 1)g_x + 9(y_k - 2)g_y}{g_x^2 + 9g_y^2}$$

När linjesökningens s_k-värde har beräknats erhålls nästa punkt i steepestdescentmetoden av $\mathbf{p}_{k+1} = \mathbf{p}_k + s_k\,\mathbf{g}_k$.

Matlabkoden för figurens steepestdescentpromenad från $x = 3.4$, $y = 1.1$ till minimipunkten ($x = 1$, $y = 2$) blir:

```
x=3.4; y=1.1;
g=-[2*(x-1) 18*(y-2)]'; gnorm=norm(g); steg=0;
while gnorm>0.0001 & steg<40
  s=-((x-1)*g(1)+9*(y-2)*g(2))/(g(1)^2+9*g(2)^2)
  x=x+s*g(1); y=y+s*g(2);
  g=-[2*(x-1) 18*(y-2)]'; gnorm=norm(g); steg=steg+1;
end
minpunkt=[x y], minvarde=(x-1)^2+9*(y-2)^2+1, antalsteg=steg
```

Om vi inte förmår att derivera $F(s)$ får vi i varje steepestdescentsteg ta till den derivatafria gyllenesnittetsökningsalgoritmen för att finna det s-värde

som minimerar $F(s)$. Sökningen behöver inte göras noggrant; det räcker med ett ungefärligt minvärde vid varje linjesökning. I nästa iteration traskar man ändå i nerförsbacke — närmare den sökta minpunkten.

Att välja startintervall för gyllenesnittetminimeringen kan vara knepigt. Det ska ju vara sådant att $F(s)$ är unimodal i intervallet. (Ta hellre ett för litet än för stort startintervall. Motivera detta!)

```
function  minex1
% Steepestdescentmetoden med gyllenesnittetsökning
  rg=(sqrt(5)-1)/2; qg=1-rg;
  x=3.4; y=1.1;
  steg=0;
  g=-fgradex(x,y); gnorm=norm(g)
  s_max=input('Linjesökningsintervall: ');      % lagom här är 0.4
  disp('     s        gnorm ')

  while gnorm>0.0001 & steg<100
    a=0; b=s_max;
    s1=a+qg*(b-a); F1=fQ(x+s1*g(1),y+s1*g(2));
    s2=a+rg*(b-a); F2=fQ(x+s2*g(1),y+s2*g(2));
    while b-a>0.001
      if F1<F2
           b=s2; s2=s1; F2=F1; s1=a+qg*(b-a); F1=fQ(x+s1*g(1),y+s1*g(2));
      else a=s1; s1=s2; F1=F2; s2=a+rg*(b-a); F2=fQ(x+s2*g(1),y+s2*g(2));
      end
    end
    s=s1;
    x=x+s*g(1);  y=y+s*g(2);
    steg=steg+1;
    g=-fgradex(x,y); gnorm=norm(g);
    disp([s gnorm])
  end
  minpunkt=[x y]
  minvarde=fQ(x,y)
  antalsteg=steg

%-------------------------------------------------------------------------
  function Q=fQ(x,y)
     Q=(x-1)^2+9*(y-2)^2+1;
%-------------------------------------------------------------------------
  function gradQ=fgradex(x,y)
     gradQ=[2*(x-1)  18*(y-2)]';
```

Steepestdescentmetoden är en säker metod för minimering, men den kräver ofta ett ansenligt antal iterationssteg innan målet är nått.

Newtons metod på gradientekvationen $\nabla Q(x_1, x_2, ..., x_n) = 0.$ ger, när den är tillämpbar, betydligt snabbare konvergens. I bilden på sidan 5 visar

de streckade räta linjerna hur bra Newtons metod fungerar vid minimering av skålfunktionen $Q(x, y) = (x-1)^2 + 9(y-2)^2 + 1$. Den tycks gå raka spåret från startpunkten till målet i minpunkten.

Vid minimering av kvadratiska uttryck som i detta fall krävs bara en iteration med Newtons metod på gradientekvationen.

Övning: Visa detta för skålfunktionen $Q(x, y)$.

2.2.3 Konjugeradegradientmetoden och BFGS-metoden

Alla minimeringsmetoder bygger på iterationsprincipen $\mathbf{p}_{k+1} = \mathbf{p}_k + s\,\mathbf{d}_k$, vilket innebär att man från punkten $\mathbf{p}_k$ väljer en sökriktning $\mathbf{d}_k$ och förflyttar sig i den riktningen fram till ett minimum vid ett visst s-värde. Därifrån upprepas förfarandet.

I steepestdescentmetoden är sökriktningen i varje steg lika med negativa gradientriktningen, alltså $\mathbf{d}_k = \mathbf{g}_k = -\nabla Q(\mathbf{p}_k)$. Det leder till en fungerande metod, tyvärr inte idealisk eftersom den iterativa lösningen i de flesta fall sicksackar sig långsamt fram mot målet i minpunkten. En liten modifiering av sökriktningen kan förhoppningsvis snabba upp konvergensen.

Konjugeradegradientmetoden är namnet på en sådan metod. Den återkommer i nästa kapitel som en betydelsefull metod för effektiv lösning av stora ekvationssystem, och där får namnet också sin förklaring. För allmänna minimeringsproblem har konjugeradegradientmetoden följande algoritm (härledd av Fletcher och Reeves, 1964).

Utgå från en startgissning $\mathbf{p}_0$, beräkna gradientvektorn och inled algoritmen med ett steepestdescentsteg: $\mathbf{d}_0 = \mathbf{g}_0 = -\nabla Q(\mathbf{p}_0), \quad \mathbf{p}_1 = \mathbf{p}_0 + s\,\mathbf{d}_0$ med s-värdet erhållet genom gyllenesnittetsökning.

I varje iterationssteg i fortsättningen bestäms sökriktningen $\mathbf{d}_k$ som en listig kombination av negativa gradientriktningen $\mathbf{g}_k$ och föregående sökriktning $\mathbf{d}_{k-1}$ enligt

$$\mathbf{d}_k = \mathbf{g}_k + \beta_k \mathbf{d}_{k-1} \quad \text{med} \quad \beta_k = \frac{\|\mathbf{g}_k\|_2^2}{\|\mathbf{g}_{k-1}\|_2^2}.$$

Nästa punkt beräknas med $\mathbf{p}_{k+1} = \mathbf{p}_k + s\,\mathbf{d}_k$, som vanligt med s-värdet erhållet via gyllenesnittetsökning. Konjugeradegradientmetoden snabbar upp konvergensen avsevärt, ofta krävs inte mer än tio iterationer där steepestdescentmetoden behöver mer än tjugofem.

BFGS-metoden är en annan fiffig algoritm för att åstadkomma bättre sökriktning och få snabb konvergens. Den utnyttjar i varje iteration en approximativ hessianmatris $\mathbf{B}_k$ och påminner därför om Newtons metod på

gradientekvationen $\nabla Q(\mathbf{p}) = \mathbf{0}$ (avsnitt 2.2.1). BFGS-metoden härstammar från 1970 då den utvecklades av Broyden, Fletcher, Goldfarb och Shanno (oberoende av varandra påstås det).

Algoritmen startar på samma sätt som konjugeradegradientmetoden, alltså med ett steepestdescentsteg. Man inför också matrisen $\mathbf{B}_0$ lika med enhetsmatrisen. I varje steg därefter, alltså för $k = 1, 2, \ldots$, utförs följande:

$$\mathbf{g}_k = -\nabla Q(\mathbf{p}_k), \quad \mathbf{y} = \mathbf{g}_k - \mathbf{g}_{k-1}, \quad \mathbf{B}_k = \mathbf{B}_{k-1} - \frac{\mathbf{y}\mathbf{y}^T}{\mathbf{y}^T\mathbf{d}_{k-1}} - \frac{\mathbf{g}_{k-1}\mathbf{g}_{k-1}^T}{\mathbf{g}_{k-1}^T\mathbf{d}_{k-1}}$$

Sökriktningen $\mathbf{d}_k$ erhålls som lösningen till ekvationssystemet $\mathbf{B}_k\mathbf{d}_k = \mathbf{g}_k$.

Med känd sökriktning går man sedan tillväga som i steepestdescentmetoden och konjugeradegradientmetoden — gyllenesnittetsökning hjälper till att finna s-värdet i den numeriska beräkningen av nästa punkt $\mathbf{p}_{k+1} = \mathbf{p}_k + s\,\mathbf{d}_k$.

Flera liknande minimeringsalgoritmer utvecklades omkring 1970, men den som varit mest lyckosam är BFGS-metoden. Algoritmen är lätt att programmera och metoden är effektiv. I praktiska experiment visar det sig att den ofta klarar sig med hälften så många iterationer som konjugeradegradientmetoden.

Det kan tilläggas att det finns en algoritm för inversen till $\mathbf{B}_k$ som kan tillämpas om man vill slippa lösa ekvationssystemet $\mathbf{B}_k\mathbf{d}_k = \mathbf{g}_k$. Börja med $\mathbf{C}_0 = \mathbf{I}$, och fortsätt i varje steg med

$$\mathbf{W} = \mathbf{I} - \frac{\mathbf{y}\mathbf{d}_{k-1}^T}{\mathbf{y}^T\mathbf{d}_{k-1}}, \quad \mathbf{C}_k = \mathbf{W}^T\mathbf{C}_{k-1}\mathbf{W} - \frac{\mathbf{d}_{k-1}\mathbf{d}_{k-1}^T}{\mathbf{y}^T\mathbf{d}_{k-1}}$$

Sökriktningen $\mathbf{d}_k$ erhålls nu ur $\mathbf{d}_k = \mathbf{C}_k\mathbf{g}_k$.

Studera Heath avsnitt 6.1, 6.2.2, 6.4, 6.5.

3 Algoritmer för lösning av glesa linjära system

En gles matris karakteriseras av att en mycket stor andel av matriselementen utgörs av nollor. Enhetsmatrisen och diagonalmatrisen är de enklaste exemplen.

3.1 Bandmatriser, speciellt tridiagonala systemmatriser

Bandmatriser är den typ av glesa matriser där alla ickenollelement finns koncentrerade i ett band kring huvuddiagonalen. Den tridiagonala matrisen är den vanligaste typen. Vid stora tridiagonala system är det mycket ineffektivt att lagra hela matrisen. Endast tre vektorer behövs – diagonalen, superdiagonalen och subdiagonalen, det innebär ett lagringsutrymme på $3N$ element i stället för N^2.

En effektiv metod för lösning av tridiagonala system är algoritmen `tridia` som finns bland kursens MATLAB-filer.[1] Studera algoritmen i NAM avsnitt 1.6.

Matriser med bandbredden fem (diagonalen samt två super- och två subdiagonaler) kommer vi att stöta på i kursavsnittet om randvärdesproblem vid differentialekvationer. I MATLAB finns effektiva `sparse`-matrisalgoritmer för alla typer av glesa matriser. Den speciella kompakta `sparse`-hanteringen av bandmatriser kan ses som en generalisering av kursens `tridia`-algoritm, men detta återkommer vi till senare.

3.2 Nästan tridiagonalt, Sherman-Morrisons algoritm

Vid differensmetoder för lösning av problem med periodiska förlopp uppkommer ekvationssystem $\mathbf{Ax} = \mathbf{b}$, där matrisen har följande struktur:

$$\begin{pmatrix} * & * & 0 & 0 & \cdots & 0 & * \\ * & * & * & 0 & \cdots & 0 & * \\ 0 & * & * & * & \cdots & 0 & * \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & * & * & * & * \\ 0 & 0 & \cdots & 0 & * & * & * \\ * & * & \cdots & * & * & * & * \end{pmatrix} \quad \text{eller på blockform:} \quad \begin{pmatrix} \mathbf{T} & \mathbf{c} \\ \mathbf{d}^T & \alpha \end{pmatrix}$$

$N{\times}N$-matrisen består alltså av en tridiagonal $(N{-}1){\times}(N{-}1)$-matris $\mathbf{T}$, kompletterad till höger med en kolumnvektor $\mathbf{c}$ och nedanför med en radvektor $\mathbf{d}^T$. I nedre högra hörnet finns en skalär storhet α.

[1] Kopiera filen `tridia.m` från kurskatalogen.

Vi vill gärna kunna använda den effektiva `tridia`-lösaren, men den går ju enbart att utnyttja på ekvationssystem $\mathbf{A}\mathbf{x} = \mathbf{b}$ med en äkta tridiagonal systemmatris. Sherman-Morrisons förfarande utnyttjar att den ursprungliga nästantridiagonala matrisen kan delas upp i de fyra storheterna i det högra uttrycket. Även högerledsvektorn behöver spjälkas upp — låt oss använda beteckningen $\mathbf{f}$ för de första $N-1$ komponenterna i vektorn $\mathbf{b}$. Den okända lösningsvektorn skrivs som en $(N-1)$-vektor $\mathbf{x}$ och en separat okänd komponent x_N.

$$\left(\begin{array}{cc} \mathbf{T} & \mathbf{c} \\ \mathbf{d}^T & \alpha \end{array}\right)\left(\begin{array}{c} \mathbf{x} \\ x_N \end{array}\right) = \left(\begin{array}{c} \mathbf{f} \\ b_N \end{array}\right) \quad \Longrightarrow \quad \left.\begin{array}{rl} \mathbf{T}\mathbf{x} + x_N\mathbf{c} & = \mathbf{f} \\ \mathbf{d}^T\mathbf{x} + x_N\alpha & = b_N \end{array}\right\}$$

Den övre ekvationen kan skrivas om som

$$\mathbf{x} = \mathbf{T}^{-1}\mathbf{f} - x_N\mathbf{T}^{-1}\mathbf{c} \quad \Rightarrow \quad \mathbf{x} = \mathbf{u} - x_N\mathbf{v}$$

där $\mathbf{u}$ erhålls med `tridia` som lösning till $\mathbf{T}\,\mathbf{u} = \mathbf{f}$ och $\mathbf{v}$ som lösning till $\mathbf{T}\,\mathbf{v} = \mathbf{c}$. Sätt in uttrycket för $\mathbf{x}$ i ekvationen $\mathbf{d}^T\mathbf{x} + x_N\alpha = b_N$, lös ut x_N:

$$\mathbf{d}^T(\mathbf{u}-x_N\mathbf{v})+x_N\alpha = b_N \;\Rightarrow\; x_N(\alpha-\mathbf{d}^T\mathbf{v}) = b_N-\mathbf{d}^T\mathbf{u} \;\Rightarrow\; x_N = \frac{b_N - \mathbf{d}^T\mathbf{u}}{\alpha - \mathbf{d}^T\mathbf{v}}$$

Nu är allting känt! Genom att x_N har beräknats, är också de första $N-1$ lösningskomponenterna kända: $\mathbf{x} = \mathbf{u} - x_N\mathbf{v}$. Två anrop av `tridia` plus några enkla operationer är allt som har behövts. Se exemplet i avsnitt 3.4.

Studera Heath avsnitt 2.4.9, 2.5, 2.7.

3.3 Konjugeradegradientmetoden på linjära system

Som alternativ till gausselimination för stora glesa och välkonditionerade ekvationssystem $\mathbf{A}\mathbf{x} = \mathbf{b}$ kan konjugeradegradientmetoden (CG) användas. Det är en iterativ metod som fungerar bäst om systemmatrisen är symmetrisk och positivt definit, dvs alla egenvärden positiva. Metoden (se avsnitt 2.2.3) bygger på att problemet kan betraktas som ett minimeringsproblem:

$$\text{Minimera } \; F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} \tag{1}$$

Lösningen till ett minimeringsproblem måste uppfylla att gradienten är noll, det vill säga $\nabla F(\mathbf{x}) = 0$. Gradienten till funktionen $F(\mathbf{x})$ kan skrivas

$$\nabla F(\mathbf{x}) = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T)\mathbf{x} - \mathbf{b}$$

Övning: Härled uttrycket för gradienten genom derivering av dubbla summauttryck.

För en symmetrisk matris då $\mathbf{A}^T = \mathbf{A}$, blir gradientuttrycket helt enkelt $\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$. En iterativ konvergerande gradientmetod för minimeringsproblemet leder därför samtidigt fram till lösningen till $\mathbf{A}\mathbf{x} - \mathbf{b} = 0$.

Enklaste gradientmetoden är steepestdescentmetoden som i varje iterationssteg ger uppmaningen: *Gå i negativa gradientriktningen ett litet stycke* — så länge som det är nedförsbacke. Gradienten är $\mathbf{A}\mathbf{x} - \mathbf{b}$; och negativa gradienten är alltså $\mathbf{b} - \mathbf{A}\mathbf{x}$, det som vi brukar kalla residualvektorn $\mathbf{r}$.

Ett steepestdescentsteg lyder $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{r}_k$ med $\alpha_k = \|\mathbf{r}_k\|^2/(\mathbf{r}_k^T \mathbf{A}\mathbf{r}_k)$.

Övning: Härled uttrycket för α_k genom att finna minimum av (1) längs sökriktningen. Det gäller alltså att via derivering med avseende på α finna det α-värde som minimerar $F(\mathbf{x}_k + \alpha\, \mathbf{r}_k)$.

Steepestdescentmetoden är simpel och kräver ofta ett stort antal iterationer, så vi önskar oss en bättre algoritm. Om man ruckar lite på sökriktningen, alltså $\mathbf{r}_k$, och i stället väljer en sökriktning $\mathbf{d}_k$ som en listigt vald linjärkombination av residualvektorn $\mathbf{r}_k$ och det förra iterationsstegets sökriktning $\mathbf{d}_{k-1}$, erhåller vi den betydligt effektivare *konjugeradegradientmetoden.*[2] Formeln för $\mathbf{d}_k$ blir

$$\mathbf{d}_k = \mathbf{r}_k + \beta_k \mathbf{d}_{k-1} \quad \text{med} \quad \beta_k = \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{r}_{k-1}\|^2}$$

Uttrycket för $\mathbf{x}_{k+1}$ blir därefter

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \quad \text{med} \quad \alpha_k = \frac{\|\mathbf{r}_k\|^2}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k}$$

Uttrycken för α_k och β_k har härletts så att metoden ger snabbast möjliga minimering längs så kallade konjugerade riktningar. Definitionsmässigt är vektorerna $\mathbf{u}$ och $\mathbf{v}$ konjugerade om $\mathbf{u}^T\mathbf{A}\mathbf{v} = 0$. Tillämpat på successiva sökriktningar uppfylls därför $\mathbf{d}_k^T \mathbf{A}\mathbf{d}_{k-1} = 0$.

Metoden är till sin fördel vid lösandet av stora glesa och välkonditionerade ekvationssystem. Vid lösning av elliptiska problem vid partiella differentialekvationer med finitadifferensmetoder (FDM) eller finitaelementmetoder (FEM) uppkommer ekvationssystem med just de här egenskaperna. Det kan då röra sig om 3D-potentialproblem med tusentals obekanta. Här är konjugeradegradientmetoden ett viktigt numeriskt verktyg.

I MATLAB kan konjugeradegradientmetoden för ett system $\mathbf{A}\mathbf{x} = \mathbf{b}$ med symmetrisk positivt definit $n \times n$-matris beskrivas med några få rader.

[2]För fullständig härledning, se Heath avsnitt 11.5.5.

```
function [x,iter]=fcgsolve(A,b,reltol)
   n=length(b); x=zeros(n,1); r=b;  rtr=r'*r;
   d=zeros(n,1); iter=0; relkoll=1;
   beta=0;
   while relkoll>reltol
     d=r+beta*d;  Ad=A*d;
     alfa=rtr/(d'*Ad);
     dx=alfa*d; x=x+dx; iter=iter+1;
     relkoll=norm(dx,inf)/norm(x,inf);
     r=r-alfa*Ad;  rtrold=rtr;  rtr=r'*r;
     beta=rtr/rtrold;
   end
```

Studera Heath avsnitt 11.5.5.

3.4 Exempel: system med nästan tridiagonal matris

Större tillämpningar av metoderna för glesa ekvationssystem kommer vi till senare i kursen. Ett exempel som anknyter till splines får illustrera användningen av metoderna för glesa system. Vid beräkning av kubiska splines för interpolation bestäms splinekurvans lutningar i interpolationspunkterna av ett ekvationssystem med tridiagonal matris.[3] Första och sista raden i systemet är beroende av villkoren för lutningarna i intervallets ändpunkter.

I det periodiska fallet kommer matrisen att förutom sin tridiagonala struktur få ett nollskilt element överst till höger och nederst till vänster. Om interpolationspunkterna är ekvidistanta får matrisen det utseende som vi nu ska betrakta närmare: De nollskilda elementen i exemplets $n \times n$-matris utgörs av fyror i alla diagonalelement, ettor i super- och subdiagonalen samt en etta i vardera övre högra och nedre vänstra hörnet.

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

Låt oss studera tre metoder för lösning av ett sådant nästan tridiagonalt system. Eftersom det är effektiviteten hos algoritmerna som är av intresse och inte lösningen i sig, så kan vi ha en godtycklig högerledsvektor.

[3] Se NAM avsnitt 3.3.2.

Först används vanlig gausselimination som ju inte är särskilt effektiv på ett stort glest system. Därefter prövas Sherman-Morrisons algoritm och till slut konjugeradegradientmetoden.

I Sherman-Morrisons blockuppdelning av matrisen kommer både vektorn **c** och vektorn **d** att bestå av nollor utom första och sista elementet som är en etta. **T** är tridiagonal och det ensamma elementet nere i högra hörnet har värdet fyra. Studera algoritmen i MATLAB-koden!

```
% Tre metoder, först A\b, sedan Sherman-Morrison och sist CG-metoden
  n=250;
  for nr=1:4             % studera fallen n=250, 500, 1000, 2000
    tic, sup=ones(n-1,1);
    A=4*eye(n)+diag(sup,1)+diag(sup,-1); A(1,n)=1; A(n,1)=1;
    b=(1:n)';
    xgauss=A\b;  fulltid(nr)=toc;

  % Sherman-Morrison med tridia
    tic
    dia=4*ones(n-1,1); sup=ones(n-2,1);
    alfa=4;  c=[1; zeros(n-3,1); 1];  d=c;
    f=b(1:n-1); bn=b(n);
    u=tridia(dia,sup,sup,f);
    v=tridia(dia,sup,sup,c);
    xn=(bn-d'*u)/(alfa-d'*v); xsh=[u-xn*v; xn];
    smtid(nr)=toc;

  % Konjugeradegradientmetoden
    tic
    [xcg,iter]=fcgsolve(A,b,1e-7);
    cgtid(nr)=toc;
    n=2*n;
  end
  disp([fulltid' smtid' cgtid'])
```

Tabellen nedan visar hur lång tid i sekunder som de olika metoderna tar. Vi kan dra slutsatsen att ju större system som ska lösas, ju mer lönar det sig att välja en metod som utnyttjar matrisens egenskaper.

n	A\b	ShermMorris	Konjgrad
250	0.06	0.025	0.016
500	0.25	0.057	0.037
1000	1.50	0.082	0.330
2000	14.80	0.190	2.210

4 Egenvärdesproblem för matriser

4.1 Egenvärden och egenvektorer

Viktiga begrepp i algebran är egenvärden och egenvektorer till matriser. Här ska vi behandla olika numeriska metoder för beräkning av dem. Men det är lämpligt att du först återvänder till läroboken i linjär algebra och friskar upp kunskapen om egenvärden, egenvektorer och deras innebörd.

Egenvärdesproblemet: Givet en $n \times n$ matris $\mathbf{A}$, finn en skalär storhet λ och en från noll skild vektor $\mathbf{x}$ sådan att $\mathbf{Ax} = \lambda\mathbf{x}$.

Storheten λ är egenvärde till matrisen och tillhörande vektor $\mathbf{x}$ är egenvektor. Sambandet kan också skrivas på formen $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$. Detta homogena ekvationssystem har en lösning skild från noll endast om determinanten för vänsterledsmatrisen är noll, alltså

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \tag{2}$$

Vänsterledet är ett n-tegradspolynom i λ. Polynomekvationen, som kallas *karakteristiska ekvationen*, har n rötter varav några kan vara komplexa och vissa kan vara sammanfallande. Detta innebär att en $n \times n$ matris har n egenvärden som i teorin bestäms av karakteristiska ekvationens rötter.

För de flesta typer av matriser finns betydligt effektivare metoder för beräkning av egenvärden, men för två matristyper är karakteristiska ekvationen (2) mycket lämplig, nämligen om $\mathbf{A}$ är diagonalmatris eller triangulär matris. Eftersom determinanten i dessa båda fall är lika med produkten av diagonalelementen, blir ekvation (2) helt enkelt $\Pi_{j=1}^{n}(a_{jj} - \lambda) = 0$, som har rötterna $\lambda_j = a_{jj},\ j = 1, ..., n$. Egenvärdena utgörs alltså av matrisens diagonalelement i fallen diagonalmatris och triangulär matris.

Övning: Visa med karakteristiska ekvationen att $\mathbf{A}^T$ har samma egenvärden som $\mathbf{A}$.

För en symmetrisk matris med reella element gäller den mycket viktiga egenskapen att alla egenvärden är reella och de n egenvektorerna är – eller kan göras – ortogonala (mer om detta längre fram).

I MATLAB kan man använda funktionen `eig(A)` för att beräkna samtliga egenvärden till en matris. Med satsen `lambda=eig(A)` erhålls en vektor innehållande egenvärdena. Med satsen `[V,D]=eig(A)` erhålls en diagonalmatris `D` med egenvärdena som diagonalelement och en matris `V` vars kolumner utgör motsvarande egenvektorer. Att utnyttja `eig`-funktionen är alldeles utmärkt för rimligt små matriser, men den kan bli mycket tidskrävande vid stora matriser. I praktiska tillämpningar är det oftast bara ett litet fåtal egenvärden

som man är intresserad av (t ex det till beloppet största eller kanske de tre till beloppet minsta egenvärdena). I sådana fall är `eig(A)` ineffektiv och vi har behov av smidigare metoder.

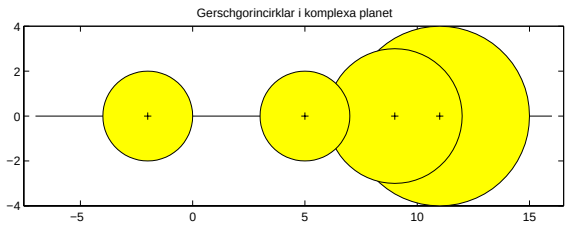
4.2 Gerschgorincirklar

Man har i vissa sammanhang nytta av att få en grov skattning av egenvärdena; det vore bra om de kunde stängas in inom begränsade områden i komplexa planet. Som tur är lyckades den ryske matematikern S.Gerschgorin 1941 bevisa ett teorem som kallas *Gerschgorins sats* och som lyder:
Varje egenvärde ligger i unionen av cirklarna

$$|\lambda - a_{jj}| \leq \sum_{k=1,k\neq j}^{n} |a_{jk}|, \quad j = 1, ..., n.$$

Gerschgorins sats säger vidare att om m cirklar har viss överlappning med varandra men inte nuddar de övriga $n - m$ cirklarna, så ligger exakt m egenvärden i unionen av dessa m cirklar.

För $\mathbf{A} = \begin{pmatrix} 9 & -2 & 1 & 0 \\ 1 & 5 & 0 & -1 \\ 0 & -1 & -2 & 1 \\ 3 & 0 & 1 & 11 \end{pmatrix}$
erhåller vi cirklarna
$|\lambda - 9| \leq 3, \ |\lambda - 5| \leq 2,$
$|\lambda + 2| \leq 2, \ |\lambda - 11| \leq 4.$



Ett egenvärde finns i den vänstra cirkeln och tre stycken finns i unionen av de övriga.

4.3 Likformighetstransformation

En $n \times n$-matris $\mathbf{B}$ sägs vara likformig med $\mathbf{A}$ om följande samband finns: $\mathbf{B} = \mathbf{P}^{-1}\mathbf{AP}$, där $\mathbf{P}$ är en ickesingulär men för övrigt godtycklig $n \times n$-matris. $\mathbf{B}$ har erhållits genom en så kallad likformighetstransformation.

Att vara likformig är en god egenskap som innebär att matriserna $\mathbf{A}$ och $\mathbf{B}$ har samma egenvärden. De har inte samma egenvektorer, men följande samband råder: låt $\mathbf{A}$ ha egenvektorn $\mathbf{x}$, då har $\mathbf{B}$ egenvektorn $\mathbf{y} = \mathbf{P}^{-1}\mathbf{x}$.
Härledning: Låt $\mathbf{B}$ ha egenvärdet μ och egenvektorn $\mathbf{y}$. Definitionsmässigt har vi då $\mathbf{By} = \mu\mathbf{y}$ som med instoppad likformighetstransformation leder till $\mathbf{P}^{-1}\mathbf{APy} = \mu\mathbf{y} \implies \mathbf{APy} = \mu\mathbf{Py}$. Men vi vet att för matrisen $\mathbf{A}$

gäller $\mathbf{Ax} = \lambda\mathbf{x}$. Identifiering ger därmed $\mu = \lambda$ och $\mathbf{Py} = \mathbf{x}$ som också kan skrivas $\mathbf{y} = \mathbf{P}^{-1}\mathbf{x}$.

Studera Heath avsnitt 4.1 och 4.2.

4.4 Potensmetoden

Potensmetoden är en iterativ numerisk metod för att beräkna det till beloppet största egenvärdet och den tillhörande egenvektorn. Principen är att man utgår från en (nästan) godtycklig vektor $\mathbf{x}_0$ och successivt bildar $\mathbf{x}_1 = \mathbf{A}\mathbf{x}_0$, $\mathbf{x}_2 = \mathbf{A}\mathbf{x}_1$ $(= \mathbf{A}^2\mathbf{x}_0)$, $\mathbf{x}_3 = \mathbf{A}\mathbf{x}_2$ $(= \mathbf{A}^3\mathbf{x}_0)$ och så vidare. Följden av vektorer konvergerar mot egenvektorn $\mathbf{x}$ till det dominerande egenvärdet. Egenvärdet kan därefter erhållas ur sambandet $\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T\lambda\mathbf{x}$, dvs $\lambda = \mathbf{x}^T\mathbf{A}\mathbf{x}/\mathbf{x}^T\mathbf{x}$ (som kallas rayleighkvoten).

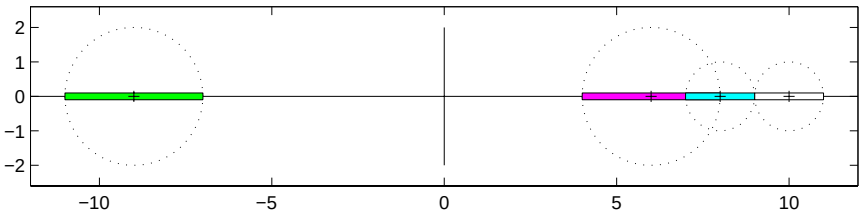
Detta är visserligen ett enkelt beräkningsförfarande men det ger snart klumpigt stora komponenter i vektorerna. Mycket bättre blir algoritmen om vektorn normeras i varje steg, antingen så att maxkomponenten normeras till ett eller vektorns längd normeras till ett. Vi väljer det senare och inför i varje iterationssteg vektorn $\mathbf{t} = \mathbf{x}_k/\|\mathbf{x}_k\|_2$, för då gäller att $\|\mathbf{t}\|_2 = 1$. Iterationen fortsätter med att vi bildar vektorn $\mathbf{x}_{k+1}$ genom att multiplicera $\mathbf{A}$ med $\mathbf{t}$. I därpå följande steg normeras $\mathbf{x}_{k+1}$ till en ny vektor $\mathbf{t}$.

I varje iteration bildar vi dessutom kvoten $q = \mathbf{t}^T\mathbf{A}\mathbf{t}/\mathbf{t}^T\mathbf{t}$, där nämnaren har värdet ett (eftersom $\mathbf{t}^T\mathbf{t} = \|\mathbf{t}\|_2^2 = 1$). Täljaren kan skrivas $\mathbf{t}^T\mathbf{x}$ (indiceringen av $\mathbf{x}$ kan skippas). Följden av q-värden konvergerar mot det dominanta egenvärdet. När differensen mellan två på varandra följande q-värden ligger inom given toleransgräns avbryts iterationerna.

För 4×4-matrisen $\mathbf{A}$ i koden nedan behövs 56 iterationer med potensmetoden innan avbrottsvillkoret $|q_n - q_{n-1}| < 10^{-5}$ är uppfyllt. Det till beloppet största egenvärdet beräknas till 10.2389.

```
A=[10  1  0  0
    1  6  0  1
    0  0  8  1
    0  1  1 -9];
x=[1 1 1 1]'; q0=0; dif=1; iter=0;
while abs(dif)>1e-5
  t=x/norm(x); x=A*t;
  q=t'*x; dif=q-q0;
  q0=q; iter=iter+1;
end
lambda=q, egenvektor=x'/norm(x), iter
```

Här ser vi gerschgorincirklarna för matrisen i MATLAB-koden. Eftersom matrisen är symmetrisk vet vi att egenvärdena är reella. Cirklarna blir därför tillplattade till intervall på reella axeln.



Matlabs `eig(A)` ger egenvärdena 10.2389, 8.0600, 5.8259 och −9.1247.

Potensmetodens konvergens beror av kvoten mellan det till beloppet näst största egenvärdet och det till beloppet största. Beteckna de successivt erhållna egenvärdesapproximationerna till λ med $q_j,\ j = 1, 2, ..., n$. Då gäller:

$$|\lambda - q_n| \approx K\,|\lambda - q_{n-1}| \quad \text{där} \quad K = \left(\frac{|\lambda|_{nextmax}}{|\lambda|_{max}}\right)^p,$$

med $p = 1$ om matrisen är osymmetrisk och $p = 2$ om den är symmetrisk. I varje iteration förbättras noggrannheten med faktorn K:

$$|\lambda - q_n| \approx K\,|\lambda - q_{n-1}| \approx K^2|\lambda - q_{n-2}| \approx ... \approx K^n\,|\lambda - q_0|.$$

Det blir långsam konvergens om kvoten K ligger strax under ett och snabb om K är nära noll.

I exemplet ovan är matrisen symmetrisk; vi får $K = (9.1247/10.2389)^2 \approx 0.793$. Efter 56 iterationer har vi $0.793^{56} \approx 2 \cdot 10^{-6}$; så mycket har felet reducerats från starten.

Ett uttryck för osäkerheten i det beräknade egenvärdet när vi känner till differensen $|q_n - q_{n-1}|$ är

$$|\lambda - q_n| \leq \frac{K}{1-K}\,|q_n - q_{n-1}|$$

Härledning:

$$|\lambda - q_n| \approx K|\lambda - q_{n-1}| = K|\lambda - q_{n-1} + q_n - q_n| = K|(\lambda - q_n) + (q_n - q_{n-1}| \leq$$
$$\leq K|\lambda - q_n| + K|q_n - q_{n-1}| \implies (1-K)\,|\lambda - q_n| \leq K|q_n - q_{n-1}|$$

Slutlig division med $(1 - K)$ leder till osäkerhetsformeln ovan.

I vårt exempel gäller vid avbrottet att $|q_n - q_{n-1}| < 10^{-5}$, och vi råkar känna till $K = 0.793$. Osäkerheten i egenvärdet är $0.793/0.207 \cdot 10^{-5} = 3.83 \cdot 10^{-5} < 0.5 \cdot 10^{-4}$; alla sex siffrorna i 10.2389 är alltså tillförlitliga.

Vi kan formulera algoritmen för potensmetoden så att den avbryter när det dominerande egenvärdet erhållits med önskad relativ tolerans.
Som approximation till K-värdet används kvoten $|q_n - q_{n-1}|/|q_{n-1} - q_{n-2}|$.

```
% Potensmetoden
  A=[10  1  0  0
      1  6  0  1
      0  0  8  1
      0  1  1 -9];
  x=[1 1 1 1]';  reldif=1;
  q0=0; dif0=1; iter=0;
  while reldif>0.5e-6
    t=x/norm(x);
    x=A*t;  q=t'*x
    dif=q-q0;
    K=abs(dif/dif0);                % uppskattning av K-värdet
    reldif=abs(K/(1-K)*dif/q);
    q0=q;
    dif0=dif; iter=iter+1;
  end
  K, lambda=q
  egenvektor=x'/norm(x), iter
```

4.5 Inversa potensmetoden — även kallad inversiteration

Med inversa potensmetoden bestämmer man det till beloppet minsta egenvärdet. Bakgrunden är att egenvärdena till inversmatrisen $\mathbf{A}^{-1}$ är $\mu = 1/\lambda$. Om man använder potensmetoden på $\mathbf{A}^{-1}$ bestäms $\mu_{maxbelopp}$, men det uppfyller ju $\mu_{maxbelopp} = 1/\lambda_{minbelopp}$. Alltså erhålls $\lambda_{minbelopp}$ genom invertering av det beräknade μ-värdet.

Potensmetodens centrala beräkningsformel är, om inversmatrisen betraktas, $\mathbf{x} = \mathbf{A}^{-1}\mathbf{t}$. Man vill undvika att invertera matriser, och det kan ordnas genom att vi skriver om formeln till $\mathbf{A}\mathbf{x} = \mathbf{t}$.

Det blir nu ett linjärt ekvationssystem som kan lösas med `x=A\t` (eller för bandmatriser med specialvarianten `tridia` eller sparsematriskommandona). Förutom denna sats och den viktiga inverteringen `lambda=1/q` är programkoden nedan identisk med potensmetodkoden.

```
% Inversa potensmetoden
  A=[10  1  0  0
      1  6  0  1
      0  0  8  1
      0  1  1 -9];
  x=[1 1 1 1]';  reldif=1;
  q0=0; dif0=1; iter=0;
```

```
while reldif>0.5e-6
  t=x/norm(x);
  x=A\t;  q=t'*x;
  dif=q-q0;
  K=abs(dif/dif0);
  reldif=abs(K/(1-K)*dif/q);
  q0=q;
  dif0=dif; iter=iter+1;
end
K, lambda=1/q
egenvektor=x'/norm(x), iter
```

23 iterationer krävs för att få det till beloppet minsta egenvärdet 5.8259.

Inversa potensmetoden med skift
Om man söker det egenvärde till $\mathbf{A}$ som ligger närmast ett givet värde s, kan man använda inversa potensmetoden efter att först ha gjort ett så kallat skift, som kan betraktas som en förskjutning av origo sträckan s.

Skapa matrisen $\mathbf{B} = \mathbf{A} - s \cdot \mathbf{I}$ som har egenvärdena $\lambda_B = \lambda_A - s$.

Med inversa potensmetoden på $\mathbf{B}$ bestäms dess till beloppet minsta egenvärde. Sedan gäller enligt sambandet ovan att $\lambda_A = s + \lambda_B$, vilket då blir det sökta egenvärdet till $\mathbf{A}$, alltså det som ligger närmast talet s.

Studera Heath avsnitt 4.5.1 – 4.5.3.

5 Tillämpningar med minstakvadratmetoden

5.1 Linjär och ickelinjär modellanpassning

Vid modellanpassning ska man, såsom namnet antyder, anpassa en modell till en uppsättning givna mätdata $(t_i, y_i),\ i = 1, 2, ..., m$. Modellfunktionen har en viss given grundform med några okända parametrar att bestämma. Några exempel på linjära modellfunktioner är $F(t) = c_1 + c_2\, t$ (rät linje), $F(t) = c_1 + c_2\, t + c_3\, t^2$ (parabel) och $F(t) = c_1 + c_2 \cos \omega t + c_3 \sin \omega t$ för ett givet ω (trigonometriskt polynom). Det allmänna utseendet hos en linjär modellfunktion är

$$F(t) = c_1\phi_1(t) + c_2\, \phi_2(t) + \ldots + c_n\, \phi_n(t) \tag{3}$$

alltså en linjärkombination av ett litet antal på förhand valda basfunktioner $\phi_k(t),\ k = 1, 2, ..., n,\ n < m$.

Det linjära modellanpassningsproblemet kan omformuleras som ett överbestämt linjärt ekvationssystem $\mathbf{Ac} \approx \mathbf{y}$, där vektorn $\mathbf{c}$ består av de okända parametrarna och högerledsvektorn $\mathbf{y}$ innehåller mätdata. Matrisen $\mathbf{A}$ byggs upp av basfunktionerna, på så sätt att kolumn k innehåller värdena $\phi_k(t_i),\ i = 1, 2, ..., m$. Avvikelsen mellan högerled och vänsterled kallas residualvektor och brukar betecknas $\mathbf{r}$, alltså $\mathbf{r} = \mathbf{y} - \mathbf{Ac}$.

Minstakvadratmetoden är den i särklass mest använda lösningsmetoden för överbestämda linjära ekvationssystem. Den så kallade minstakvadratlösningen $\mathbf{c}$ har egenskapen att minimera den euklidiska normen för residualvektorn, alltså $\|\mathbf{y} - \mathbf{Ac}\|_2$. Ofta är det praktiskt att i stället betrakta felkvadratsumman $\mathbf{r}^T\mathbf{r} = r_1^2 + r_2^2 + ... + r_m^2$, som är detsamma som kvadraten på den euklidiska normen. En annan formulering är därför ”Vid minstakvadratmetoden minimeras felkvadratsumman”.

Studera NAM avsnitt 2.1 – 2.5 för repetition av minstakvadratmetoden och normalekvationerna $\mathbf{A}^T\mathbf{Ac} = \mathbf{A}^T\mathbf{y}$.

Ickelinjär modellanpassning

Vid ickelinjär modellanpassning är modellfunktionens form sådan att de okända parametrarna ingår ickelinjärt. $F(t) = c_1/(1 + c_2\, e^{-c_3\, t})$ är ett exempel på en ickelinjär modell.

Det ickelinjära modellanpassningsproblemet kan omformuleras som ett överbestämt ickelinjärt ekvationssystem, som sedan i sin tur kan lösas iterativt med Gauss-Newtons metod. I varje iteration uppstår ett överbestämt linjärt system på vilket minstakvadratmetoden tillämpas. Studera detta mer detaljerat i NAM avsnitt 6.10.

Läs NAM kapitel 2 samt avsnitt 6.10.
Läs Heath avsnitt 3.1, 3.2 och 6.6.

5.2 Minstakvadratanpassning med linjära och kubiska splines

Vid interpolationsproblem med kubiska splines används hermiteinterpolationsformeln som ansats för de styckvisa tredjegradspolynomen. Ett tridiagonalt ekvationssystem måste lösas för att åstadkomma de rätta lutningarna vid varje styckes ändpunkter (se NAM avsnitt 3.3.2).

Kubiska splines kan i vissa sammanhang vara lämplig modellfunktion vid minstakvadratanpassning till ett stort antal mätpunkter (x_i, y_i). Som basfunktioner i uttrycket (3) lämpar sig så kallade bellsplines[4] bäst.

Hela intervallet delas upp i ett antal stycken, och vi betecknar x-värdena som avgränsar styckena med $T_1, T_2, \ldots, T_N$. Dessa T_k-värden brukar kallas knutpunkter. Vid varje knutpunkt sker polynombyte (vid interpolation sammanfaller knutpunkterna med interpolationspunkternas x-koordinater).

5.2.1 Approximation med linjära splines

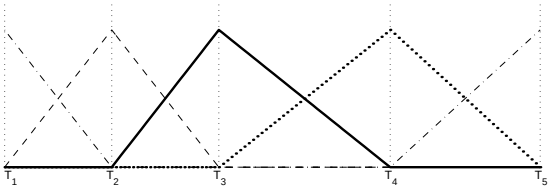
Innan vi går vidare till kubiska bellsplines ska vi studera minstakvadratapproximation med linjära splines. Med detta menar vi styckvis linjära funktioner, alltså räta linjer som byts vid varje knutpunkt men bildar en kontinuerlig polygonlinje. Modellfunktionen skrivs som

$$F(x) = c_1 H_1(x) + c_2 H_2(x) + c_3 H_3(x) + \cdots + c_N H_N(x)$$

där $H_k(x),\ k = 1, 2, ..., N,$ är basfunktioner med kineshattform — så kallade hattfunktioner (även benämnda triangelfunktioner). $H_k(x)$ är noll utanför intervallet $T_{k-1} < x < T_{k+1}$. Hattfunktionen ökar från noll vid $x = T_{k-1}$ linjärt till ett vid $x = T_k$ och sjunker linjärt till noll vid $x = T_{k+1}$.

Första hattfunktionen är en halv hatt, $H_1(x) = (T_2 - x)/(T_2 - T_1)$, som sjunker från ett vid $x = T_1$ till noll vid $x = T_2$ och är noll därefter.

Sista basfunktionen $H_N(x)$ är noll fram till $x = T_{N-1}$, därefter växer den enligt $H_N(x) = (x - T_{N-1})/(T_N - T_{N-1})$.



[4]Heath använder beteckningen B-splines.

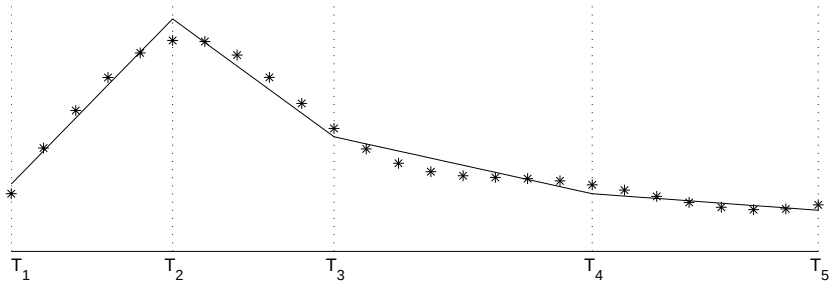
För de övriga hattfunktionerna ($k = 2, ..., N-1$) gäller:

$$H_k(x) = \begin{cases} (x - T_{k-1})/(T_k - T_{k-1}), & T_{k-1} \le x \le T_k \\ (T_{k+1} - x)/(T_{k+1} - T_k), & T_k \le x \le T_{k+1} \end{cases}$$

```
function f=fihatt(k,x,T)        % T är en vektor med knutpunkter
  % För hatt nummer k beräknas värdena för givna x
    h=diff(T); N=length(T); f=zeros(size(x));
    if k>1, I=find(x>=T(k-1) & x<=T(k)); f(I)=(x(I)-T(k-1))/h(k-1); end
    if k<N, I=find(x>=T(k) & x<=T(k+1)); f(I)=(T(k+1)-x(I))/h(k); end
```

När vi skriver minstakvadratproblemet på formen $\mathbf{Ac} \approx \mathbf{y}$ bildas kolumn k i matrisen $\mathbf{A}$ av de givna x_i-värdena insatta i $H_k(x)$. Det betyder att matrisen får många nollor eftersom hattfunktionen är noll utanför ett litet område. Normalekvationerna $\mathbf{A^T Ac} = \mathbf{A^T y}$ bildar ett tridiagonalt system (förvissa dig om det).

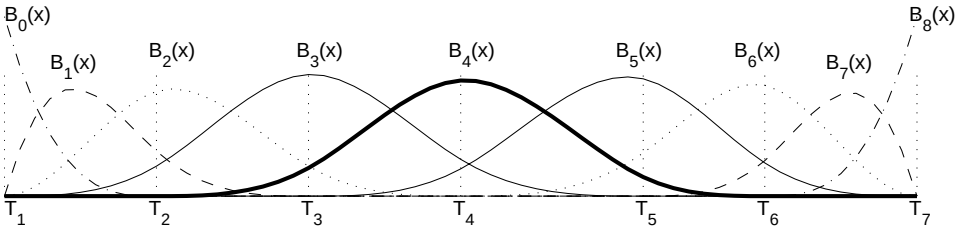
Lösning av detta system ger minstakvadratlösningen som består av koefficienterna c_1, $c_2, \ldots, c_N$ i modellfunktionen $F(x)$. Funktionskurvan är en bruten linje, en polygonkurva. $F(x)$ är kontinuerlig överallt men derivatan är diskontinuerlig vid knutpunkterna. I fallet med linjära splines gäller att c_k-värdet också utgör y-värdet vid knutpunkt nummer k.
Övning: Bevisa detta.



Här har en bruten linje med fem knutpunkter anpassats till 26 givna mätdata.

5.2.2 Approximation med bellsplines

Nu ska vi pröva en modellfunktion som i stället för räta linjer är uppbyggd av styckvisa tredjegradspolynom. Med kubiska splines kan man åstadkomma att den resulterande funktionskurvan får kontinuerlig andraderivata överallt. Kubiska splines kan byggas upp med olika ansatser. Vid modellanpassning passar det bäst med bellsplines, som påminner om kineshattarna men med mjukare blåklockeform. Basfunktionerna i modellfunktionen (3) utgörs nu av bellsplinefunktionerna $B_k(x)$.



De matematiska uttrycken för $B_k(x)$ är inte helt enkla. Om knutpunkterna är ekvidistanta, med samma inbördes avstånd $h = T_{k+1} - T_k$, kan bellsplinesuttrycken skrivas explicit, detta behandlas i nästa avsnitt.

I det allmänna fallet erhålls bellsplinefunktionen $B_k(x)$ genom följande rekursiva algoritm:

Definiera $B_k^{(0)}(x) = 1,\ T_k \le x < T_{k+1},$ och $B_k^{(0)}(x) = 0$ annars.
Fortsätt för $j = 1,\ 2,\ 3$ med

$$B_k^{(j)}(x) = (x - T_k)\frac{B_k^{(j-1)}(x)}{T_{k+j} - T_k} + (T_{k+j+1} - x)\frac{B_{k+1}^{(j-1)}(x)}{T_{k+j+1} - T_{k+1}}.$$

När $j = 3$ är bellsplinefunktionen bestämd och den övre indiceringen kan slopas. Vid vänstra och högra änden av approximationsintervallet måste åtgärder vidtas. I fallet icke-ekvidistanta knutpunkter går det till så att man inför tre nya knutpunkter som sammanfaller med T_1, alltså $T_{-2} = T_{-1} = T_0 = T_1$, och på samma sätt tre nya knutpunkter vid högra änden: $T_N = T_{N+1} = T_{N+2} = T_{N+3}$. (De totalt $N + 6$ knutpunkterna numreras tillfälligt om från 0 till N+5 för att den rekursiva algoritmen ovan ska vara giltig.)

Blåklockekurvorna $B_k(x)$ är positiva i intervallet $T_{k-2} < x < T_{k+2}$ och noll för övrigt. När man har N givna knutpunkter är antalet bellsplinefunktioner $N + 2$. Längst till vänster finns $B_0(x)$ som är noll då $x \ge T_2$, och längst till höger finns $B_{N+1}(x)$ som är noll då $x \le T_{N-1}$ (se figuren ovan).

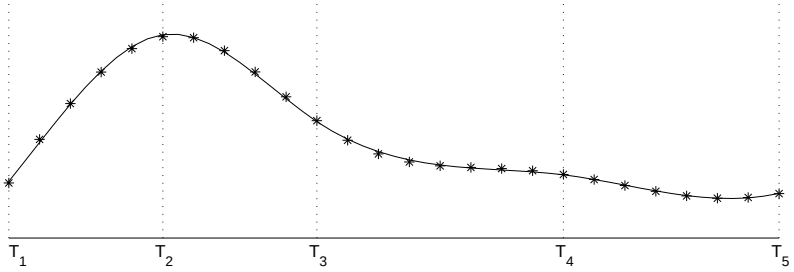
Nu gäller det att lösa approximationsproblemet att till en given uppsättning mätdata $(x_i,\ y_i),\ i = 1, 2, ..., m,$ finna en kubisk splinekurva som ger bästa möjliga anpassning. Modellfunktionen är nu en linjärkombination av bellsplines enligt

$$F(x) = c_1 B_0(x) + c_2 B_1(x) + c_3 B_2(x) + \cdots + c_{N+2} B_{N+1}(x).$$

Minstakvadratmetoden kan tillämpas på samma sätt som beskrevs för linjära splines. Det finns $N+2$ okända koefficienter att bestämma.

Problemet skrivs som vanligt på matrisform $\mathbf{Ac} \approx \mathbf{y}$, matrisen $\mathbf{A}$ har m rader och $N+2$ kolumner. Kolumn k är uppbyggd av $B_{k-1}(x_i)$ med en hel del nollelement eftersom funktionen är noll utanför ett begränsat intervall.

I MATLAB löses det överbestämda systemet med `c=A\y`. Därefter återstår bara att rita den erhållna splineskurvan $F(x)$ och notera att den ger god anpassning till de givna mätvärdena.



Figuren visar modellanpassning med bellsplines. Det är samma knutpunkter och samma 26 mätdata som vid approximationen med bruten linje i avsnitt 5.2.1.

```
function B=fbell(x,T)
 % T är kolumnvektor med knutpunkter, x är kolumnvektor med x-värden
 % B är matris med bellsplineskolumner, allmänna fallet
   m=length(x);  N=length(T);  epsi=1e-14;
   a=[T(1)*[1 1 1]'; T; T(N)*(1+epsi)*[1 1 1]'];  % N+6 knutpunkter
   n=N+5; C=zeros(m,n);
   for k=1:n
     I=find(x>=a(k) & x<a(k+1)); if ~isempty(I), C(I,k)=1; end
   end
   for j=1:3
     B=zeros(m, n-j);
     for k=1:n-j
       d1=(a(k+j)-a(k)); if abs(d1)<=epsi, d1=1; end;
       d2=(a(k+j+1)-a(k+1)); if abs(d2)<=epsi, d2=1; end;
       B(:,k)=(x-a(k)).*C(:,k)/d1 + (a(k+j+1)-x).*C(:,k+1)/d2;
     end
     C=B;
   end
```

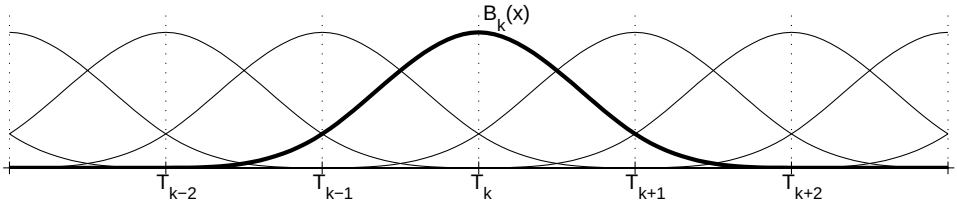
5.2.3 Approximation med ekvidistanta bellsplines

När knutpunkterna T_k ligger på inbördes lika avstånd, $h = T_{k+1} - T_k$, kan bellsplinefunktionerna få enklare explicita uttryck än allmänna fallets rekursiva uttryck. Förutom knutpunkterna $T_1, T_2, \ldots, T_N$ behövs tre spökknutpunkter till vänster och tre till höger. I förra avsnittet lät vi dem sammanfalla med T_1 respektive T_N. I det ekvidistanta fallet placeras de vänstra istället successivt på avståndet h bakåt, alltså $T_0 = T_1 - h$, $T_{-1} = T_1 - 2h$, $T_{-2} = T_1 - 3h$. På höger sida görs på motsvarande sätt:

$$T_{N+1} = T_N + h, \;\; T_{N+2} = T_N + 2h, \;\; T_{N+3} = T_N + 3h.$$

I de fyra intervall där den klockformade bellsplinekurvan är skild från noll gäller följande uttryck för $B_k(x)$:

$$\begin{array}{lll}
T_{k-2} \le x \le T_{k-1}: & t = \frac{1}{h}(x - T_{k-2}), & B_k(x) = \frac{1}{6}t^3 \\
T_{k-1} \le x \le T_k: & t = \frac{1}{h}(x - T_{k-1}), & B_k(x) = \frac{1}{6} + \frac{1}{2}(t + t^2 - t^3) \\
T_k \le x \le T_{k+1}: & t = \frac{1}{h}(T_{k+1} - x), & B_k(x) = \frac{1}{6} + \frac{1}{2}(t + t^2 - t^3) \\
T_{k+1} \le x \le T_{k+2}: & t = \frac{1}{h}(T_{k+2} - x), & B_k(x) = \frac{1}{6}t^3
\end{array}$$



Med dessa beteckningar och med den uppsättning av givna ekvidistanta knutpunkter $T_1, \ldots, T_N$ (plus införda spökknutpunkter) kommer modellfunktionen $F(x)$ att vara en linjärkombination av de $N + 2$ bellsplinefunktionerna $B_k(x)$ med index 0, 1, ..., $N+1$, alltså

$$F(x) = c_1 B_0(x) + c_2 B_1(x) + \ldots + c_{N+2} B_{N+1}(x).$$

Här följer MATLAB-kod för funktionen `fbelleq` som skapar $B_k(x)$-värden för givet k (heltal mellan 0 och $N+1$) och givna x-värden.

```
function f=fbelleq(k,x,T_1,T_N,N)
% N är antal knutpunkter (ekvidistanta fallet)
% T_1 och T_N är första och sista knutpunkt
% k är heltal mellan 0 och N+1 (numret hos önskad bellspline)
  h=(T_N-T_1)/(N-1);
  a=T_1-3*h : h :T_N+3*h;  c=1/6;
  f=zeros(size(x));  i=k+3;
  I=find(x>=a(i-2) & x<a(i-1)); t=(x(I)-a(i-2))/h; f(I)=c*t.^3;
  I=find(x>=a(i-1) & x<a(i));   t=(x(I)-a(i-1))/h; f(I)=c+(t+t.^2-t.^3)/2;
  I=find(x>=a(i) & x<a(i+1));   t=(a(i+1)-x(I))/h; f(I)=c+(t+t.^2-t.^3)/2;
  I=find(x>=a(i+1) & x<a(i+2)); t=(a(i+2)-x(I))/h; f(I)=c*t.^3;
```

Studera Heath avsnitt 7.4.3.

6 QR och SVD med tillämpningar

6.1 Ortogonalmatriser och QR-faktorisering

6.1.1 Ortogonalmatriser

En kvadratisk matris $\mathbf{Q}$ är ortogonal om $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ och $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$.
För en ortogonal matris gäller $\mathbf{Q}^T = \mathbf{Q}^{-1}$. Kolumnerna i $\mathbf{Q}$ är ortogonala mot varandra och har längden 1. Samma egenskap gäller för raderna i $\mathbf{Q}$.

Det gäller att $\|\mathbf{Qx}\|_2 = \|\mathbf{x}\|_2$, dvs en transformation med en ortogonalmatris behåller längden.
Bevis: $\|\mathbf{Qx}\|_2^2 = \mathbf{x}^T\mathbf{Q}^T\mathbf{Qx} = \mathbf{x}^T\mathbf{x} = \|\mathbf{x}\|_2^2$.

6.1.2 Householdermatriser

En användbar ortogonalmatris är den symmetriska householdermatrisen

$$\mathbf{H} = \mathbf{I} - 2\,\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}, \quad \text{där } \mathbf{v} \text{ är en väl vald vektor.}$$

Det är lätt att visa att $\mathbf{H}^T = \mathbf{H}$, dvs att matrisen är symmetrisk. (Gör det!)
Är $\mathbf{H}$ en ortogonalmatris? Visa att $\mathbf{H}^T\mathbf{H} = \mathbf{I}$.

$$\mathbf{H}^T\mathbf{H} = \mathbf{H}\,\mathbf{H} = \left(\mathbf{I} - 2\,\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\right)\left(\mathbf{I} - 2\,\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\right) = \mathbf{I} - 4\,\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}} + 4\frac{\mathbf{v}(\mathbf{v}^T\mathbf{v})\mathbf{v}^T}{(\mathbf{v}^T\mathbf{v})(\mathbf{v}^T\mathbf{v})} = \mathbf{I}$$

Problem: Hitta en matris $\mathbf{P}$ som transformerar en given vektor $\mathbf{a}$ till given vektor $\mathbf{b}$ med samma längd.

Lösning: Eftersom längden ska bevaras måste $\mathbf{P}$ vara en ortogonalmatris. Utnyttja householdermatris $\mathbf{P} = \mathbf{H} = \mathbf{I} - 2\,(\mathbf{v}\mathbf{v}^T)/(\mathbf{v}^T\mathbf{v})$, det gäller att finna vektorn $\mathbf{v}$ uttryckt i de givna $\mathbf{a}$ och $\mathbf{b}$.

$$\mathbf{b} = \mathbf{Ha} = \mathbf{a} - 2\,\frac{\mathbf{v}\mathbf{v}^T\mathbf{a}}{\mathbf{v}^T\mathbf{v}} = \mathbf{a} - \mathbf{v}\,\frac{2\,\mathbf{v}^T\mathbf{a}}{\mathbf{v}^T\mathbf{v}} = \mathbf{a} - c\,\mathbf{v} \implies \mathbf{v} = (\mathbf{a} - \mathbf{b})/c.$$

Sätt $c = 1$, det divideras i alla fall bort i uttrycket för $\mathbf{P}$. Svaret är alltså: $\mathbf{P} = \mathbf{I} - 2\,(\mathbf{v}\mathbf{v}^T)/(\mathbf{v}^T\mathbf{v})$ där $\mathbf{v} = \mathbf{a} - \mathbf{b}$.

Vanligt önskemål är att transformera en given vektor $\mathbf{a}$ till en vektor som innehåller nollor överallt utom i första elementet b_1. Eftersom vektorlängden ska bibehållas gäller alltså antingen $b_1 = \|\mathbf{a}\|_2$ eller $b_1 = -\|\mathbf{a}\|_2$.

Den här typen av transformation behövs i alla steg i QR-faktoriseringen av en matris. Tecknet för komponenten b_1 väljs då tvärtemot tecknet på a_1, alltså $b_1 = -\text{sign}(a_1)\,\|\mathbf{a}\|_2$. (Om $a_1 = 0$ tar vi $b_1 = \|\mathbf{a}\|_2$.)

6.1.3 QR-faktorisering

QR-faktoriseringen till en $m \times n$ matris $\mathbf{A}$ definieras som $\mathbf{A} = \mathbf{QR}$, där $\mathbf{Q}$ är en ortogonal $m \times m$ matris och $\mathbf{R}$ är en $m \times n$ matris vars övre del $(n \times n)$ är en triangulär matris $\mathbf{R}_n$ och elementen därunder är nollor. Matriserna $\mathbf{Q}$ och $\mathbf{R}$ byggs upp genom en följd av householdertransformationer.[5]

I MATLAB erhålls QR-faktoriseringen med satsen `[Q,R]=qr(A)`.

6.1.4 QR på överbestämda ekvationssystem

Anta nu att QR-faktoriseringen av $\mathbf{A}$ är känd. Låt $\mathbf{y}$ vara en given vektor med m komponenter. Minstakvadratlösningen till det överbestämda ekvationssystemet $\mathbf{Ac} \approx \mathbf{y}$ erhålls genom minimering av felkvadratsumman $\|\mathbf{y} - \mathbf{Ac}\|_2^2$ som vi kan manipulera med på följande sätt, där vi utnyttjar att multiplikation med en ortogonalmatris bibehåller längden.

$$\|\mathbf{y} - \mathbf{Ac}\|_2^2 = \|\mathbf{y} - \mathbf{QRc}\|_2^2 = \|\mathbf{Q}^T(\mathbf{y} - \mathbf{QRc})\|_2^2 = \|\mathbf{Q}^T\mathbf{y} - \mathbf{Rc}\|_2^2 = \|\mathbf{g} - \mathbf{Rc}\|_2^2$$

Eftersom $\mathbf{y}$ och $\mathbf{Q}$ är kända är $\mathbf{g} = \mathbf{Q}^T\mathbf{y}$ en känd vektor med m element.

Låt de första n komponenterna i vektorn $\mathbf{g}$ betecknas $\mathbf{g}_n$ och resten, dvs komponenterna $g_{n+1}, \ldots, g_m$, betecknas $\mathbf{g}_*$. Det leder till att felkvadratsumman kan utvecklas vidare:

$$\|\mathbf{g} - \mathbf{Rc}\|_2^2 = \left\| \begin{pmatrix} \mathbf{g}_n \\ \mathbf{g}_* \end{pmatrix} - \begin{pmatrix} \mathbf{R}_n\mathbf{c} \\ \mathbf{0} \end{pmatrix} \right\|_2^2 = \|\mathbf{g}_n - \mathbf{R}_n\mathbf{c}\|_2^2 + \|\mathbf{g}_* - \mathbf{0}\|_2^2.$$

Vi vill åstadkomma att felkvadratsumman ska bli så liten som möjligt. Den sista termen $\|\mathbf{g}_*\|_2^2$ kan vi inte göra någonting åt — den är oberoende av den sökta lösningsvektorn $\mathbf{c}$. Men den första termen, alltså $\|\mathbf{g}_n - \mathbf{R}_n\mathbf{c}\|_2^2$, kan den bli noll? Ja, det blir den om vi löser ekvationssystemet $\mathbf{R}_n\mathbf{c} = \mathbf{g}_n$.

Alltså, lösning av detta triangulära system ger minstakvadratlösningen $\mathbf{c}$ till problemet, för då kommer felkvadratsumman att anta sitt minsta möjliga värde som är $\|\mathbf{g}_*\|_2^2 = g_{n+1}^2 + g_{n+2}^2 + \ldots + g_m^2$.

Studera Heath avsnitt 3.4.3–3.4.5, 3.5.1.

[5] Ett litet exempel visas under kursen.

6.2 Singulärvärdesfaktorisering

Singulärvärdesfaktoriseringen (singular value decomposition, SVD) till en $m \times n$ matris $\mathbf{A}$ definieras som $\mathbf{A} = \mathbf{USV}^T$, där $\mathbf{U}$ är en ortogonal $m \times m$-matris, $\mathbf{V}$ är ortogonal $n \times n$-matris och $\mathbf{S}$ är en $m \times n$-matris uppbyggd av en diagonalmatris och en matris av nollor.

Diagonalmatrisens diagonalelement kallas singulärvärden, de är positiva och uppfyller $s_1 \geq s_2 \geq s_3 \geq \ldots \geq s_k \geq 0$, där k är det minsta av talen m och n.

Om $m > n$ ligger nollmatrisen under $n \times n$-diagonalmatrisen som betecknas $\mathbf{S}_n$ (sista elementet är singulärvärdet s_n), och om $m < n$ finns nollmatrisen till höger om diagonalmatrisen $\mathbf{S}_m$.

Singulärvärdesfaktorisering av en kvadratisk matris gjordes redan för över hundra år sedan av matematikerna Beltrami 1873 och Jordan 1874 oberoende av varandra. Namnet tillkom på 1930-talet då Eckart och Young utvidgade singulärvärdesfaktoriseringen till att vara giltig även för rektangulära matriser. Men det dröjde ytterligare drygt trettio år innan man fann att SVD kunde vara ett mycket användbart redskap vid numeriskt beräkningsarbete. Banbrytaren här var Gene Golub som med sofistikerad numerisk teknik lyckades utveckla effektiva algoritmer för beräkning av SVD-matriserna $\mathbf{U}$, $\mathbf{V}$ och $\mathbf{S}$ och också för ett otal tillämpningar av SVD.

I MATLAB erhålls singulärvärdesfaktoriseringen av matrisen $\mathbf{A}$ med satsen `[U,S,V]=svd(A)`.

6.2.1 SVD på överbestämda ekvationssystem

Vid överbestämda system gäller att $m > n$. Minstakvadratlösningen till $\mathbf{Ax} \approx \mathbf{y}$ erhålls genom minimering av felkvadratsumman $\|\mathbf{y} - \mathbf{Ax}\|_2^2$ som kan skrivas om på följande sätt:

$$\|\mathbf{y} - \mathbf{Ax}\|_2^2 = \|\mathbf{y} - \mathbf{USV}^T\mathbf{x}\|_2^2 = \|\mathbf{U}^T(\mathbf{y} - \mathbf{USV}^T\mathbf{x})\|_2^2 =$$

$$= \|\mathbf{U}^T\mathbf{y} - \mathbf{SV}^T\mathbf{x}\|_2^2 = \|\mathbf{d} - \mathbf{Sz}\|_2^2$$

där $\mathbf{d} = \mathbf{U}^T\mathbf{y}$ är en känd vektor med m element och där vi på den okända vektorn $\mathbf{x}$ gjort substitutionen $\mathbf{z} = \mathbf{V}^T\mathbf{x}$. Vektorn $\mathbf{z}$ har n komponenter.

Vi gör motsvarande förfarande som vid QR-faktoriseringen. Låt de första n komponenterna i vektorn $\mathbf{d}$ betecknas $\mathbf{d}_n$ och resten, dvs komponenterna $d_{n+1}, \ldots, d_m$, betecknas $\mathbf{d}_*$. Felkvadratsumman kan nu skrivas:

$$\|\mathbf{d} - \mathbf{Sz}\|_2^2 = \left\| \begin{pmatrix} \mathbf{d}_n \\ \mathbf{d}_* \end{pmatrix} - \begin{pmatrix} \mathbf{S}_n\mathbf{z} \\ \mathbf{0} \end{pmatrix} \right\|_2^2 = \|\mathbf{d}_n - \mathbf{S}_n\mathbf{z}\|_2^2 + \|\mathbf{d}_*\|_2^2.$$

Vid multiplikation av diagonalmatrisen $\mathbf{S}_n$ med $\mathbf{z}$ erhålls en vektor med komponenterna $s_1z_1,\ s_2z_2, \ldots, s_nz_n$.
Felkvadratsumman blir alltså $(d_1 - s_1z_1)^2 + (d_2 - s_2z_2)^2 + \ldots + (d_n - s_nz_n)^2 + d_{n+1}^2 + d_{n+2}^2 + \ldots + d_m^2$. Om vi bestämmer alla z_i-värden så att $z_i = d_i/s_i$ så åstadkommer vi att felkvadratsumman antar sitt minsta möjliga värde.

När vektorn $\mathbf{z}$ nu är beräknad, gäller det att också finna den sökta $\mathbf{x}$-vektorn. Mellan de båda vektorerna finns sambandet $\mathbf{z} = \mathbf{V}^T\mathbf{x}$. Multiplicera båda sidor med den ortogonala matrisen $\mathbf{V}$; direkt erhålls $\mathbf{x} = \mathbf{V}\mathbf{z}$, som är minstakvadratlösningen till problemet.

6.2.2 SVD på underbestämda ekvationssystem

Vid underbestämda ekvationssystem gäller att $m < n$, och då finns ingen entydig lösning till $\mathbf{A}\mathbf{x} \approx \mathbf{y}$. Med singulärvärdesfaktorisering kan vi lösa underbestämda system, som uppfyller det speciella önskemålet att lösningsvektorn $\mathbf{x}$ ska ha så liten norm som möjligt (alltså uppföra sig snällt utan stora variationer i komponenterna). Vi studerar felkvadratsumman som enligt ovan kan skrivas

$$\|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{d} - \mathbf{S}\mathbf{z}\|_2^2$$

där $\mathbf{d} = \mathbf{U}^T\mathbf{y}$ är en känd vektor med m komponenter och $\mathbf{z} = \mathbf{V}^T\mathbf{x}$ har n komponenter med $n > m$.
Matris-vektorprodukten $\mathbf{S}\mathbf{z}$ får m komponenter som är $s_1z_1,\ s_2z_2, \ldots, s_mz_m$.
Felkvadratsumman blir $(d_1 - s_1z_1)^2 + (d_2 - s_2z_2)^2 + \ldots + (d_m - s_mz_m)^2$, och den blir noll om komponenterna z_i uppfyller $z_i = d_i/s_i$ för $i = 1,\ 2,\ \ldots, m$.

Vi vet att vektorn $\mathbf{z}$ har n komponenter, men endast de första m värdena har kunnat bestämmas. Vilka värden ska $z_{m+1}, \ldots, z_n$ ha?

Här kommer önskemålet om en minstanorm-lösning in. Man vill göra euklidiska normen av lösningen $\mathbf{x}$ så liten som möjligt.

$$\|\mathbf{x}\|_2^2 = \|\mathbf{V}\mathbf{z}\|_2^2 = \|\mathbf{z}\|_2^2 = \left(\frac{d_1}{s_1}\right)^2 + \left(\frac{d_2}{s_2}\right)^2 + \ldots + \left(\frac{d_m}{s_m}\right)^2 + z_{m+1}^2 + \ldots + z_n^2.$$

Genom att sätta alla komponenterna $z_{m+1}, \ldots, z_n$ till noll får vi den önskade minstanormegenskapen hos lösningen $\mathbf{x} = \mathbf{V}\mathbf{z}$.

6.2.3 Trunkerad singulärvärdesfaktorisering

Det är ofta önskvärt att vid illakonditionerade ekvationssystem (vare sig de är över- eller underbestämda) försöka hitta en snällare lösning, det vill säga

en med mindre fluktuerande lösningskomponenter. Med trunkerad singulärvärdesfaktorisering åstadkommer man detta. Felkvadratsumman har tidigare visats bestå av summan

$(d_1 - s_1 z_1)^2 + (d_2 - s_2 z_2)^2 + \ldots + (d_n - s_n z_n)^2 + d_{n+1}^2 + d_{n+2}^2 + \ldots + d_m^2$,

och det gäller att $z_i = d_i / s_i$. När alla singulärvärden tas med blir de första n termerna noll, och felkvadratsumman blir $d_{n+1}^2 + d_{n+2}^2 + \ldots + d_m^2$.
Om vi bestämmer oss för att bara ta med r singulärvärden, så sätts z_i till noll för alla $i > r$. Felkvadratsumman kommer nu att innehålla termerna

$d_{r+1}^2 + d_{r+2}^2 + \ldots + d_{n+1}^2 + d_{n+2}^2 + \ldots + d_m^2$,

vi får alltså acceptera en större felkvadratsumma än den minimala.
Nå, blir lösningen snällare, blir dess norm mindre i det trunkerade fallet än om alla singulärvärden tas med? För att se det måste vi uttrycka matrisen $\mathbf{V}$ i sina kolumner $\mathbf{v}_i,\ i = 1, 2, \ldots, n$. Nu kan lösningsvektorn $\mathbf{x}$ skrivas

$$\mathbf{x} = \mathbf{V}\mathbf{z} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n) \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_r \\ 0 \\ 0 \end{pmatrix} = z_1\mathbf{v}_1 + z_2\mathbf{v}_2 + \ldots + z_r\mathbf{v}_r =$$

$$= \frac{d_1}{s_1}\mathbf{v}_1 + \frac{d_2}{s_2}\mathbf{v}_2 + \ldots + \frac{d_r}{s_r}\mathbf{v}_r,$$

$$\|\mathbf{x}\|_2^2 = \|\mathbf{V}\mathbf{z}\|_2^2 = \|\mathbf{z}\|_2^2 = \left(\frac{d_1}{s_1}\right)^2 + \left(\frac{d_2}{s_2}\right)^2 + \ldots + \left(\frac{d_r}{s_r}\right)^2$$

Vid trunkerad SVD blir lösningsvektorns norm som önskat mindre än om alla n termerna finns med. Hur många singulärvärden som bör medtagas är oftast svårt att förutsäga, det bör man experimentellt pröva sig fram till.

6.2.4 Minimalegenskap

Givet en matris $\mathbf{A}$, hitta den vektor $\mathbf{w}$ med längden ett, $\|\mathbf{w}\|_2 = 1$, som minimerar $\|\mathbf{A}\mathbf{w}\|_2$.
För en symmetrisk kvadratisk matris är minimivärdet lika med det till beloppet minsta egenvärdet och vektorn $\mathbf{w}$ är den normerade egenvektorn till detta egenvärde. För en $m \times n$-matris är minimivärdet lika med minsta singulärvärdet s_n och vektorn $\mathbf{w}$ är sista kolumnen i matrisen $\mathbf{V}$, alltså $\mathbf{w} = \mathbf{v}_n$.
Vi ska visa det sista påståendet.

$$\|\mathbf{A}\mathbf{w}\|_2^2 = \|\mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{w}\|_2^2 = \|\mathbf{S}\mathbf{V}^T\mathbf{w}\|_2^2 = \|\mathbf{S}\mathbf{y}\|_2^2 = (s_1 y_1)^2 + (s_2 y_2)^2 + \cdots + (s_n y_n)^2,$$

där vi infört $\mathbf{y} = \mathbf{V}^T\mathbf{w}$ som uppfyller $\|\mathbf{y}\|_2 = 1$.

Eftersom singulärvärdena är monotont avtagande gäller följande olikhet: $\sum_{k=1}^{n}(s_k y_k)^2 \geq \sum_{k=1}^{n}(s_n y_k)^2 = s_n^2 \sum_{k=1}^{n} y_k^2 = s_n^2$, vilket utgör en undre gräns. Men om man väljer $\mathbf{y}^T = (0, 0, 0, \ldots, 1)$ nås den undre gränsen. Den sökta vektorn $\mathbf{w}$ blir därmed

$$\mathbf{w} = \mathbf{V}\mathbf{y} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n) \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix} = \mathbf{v}_n.$$

6.2.5 SVD för datakomprimering

En digitaliserad bild (t ex av ett foto) kan betraktas som en matris där varje element i matrisen representerar en svärtningsgrad. Om bilden är mycket informationsrik behövs god upplösning, vilket innebär att bildmatrisen blir mycket stor. Det finns många mer eller mindre avancerade komprimeringsalgoritmer för att åstadkomma acceptabel bildkvalitet utan att hela matrisen behöver lagras.

Singulärvärdesfaktorisering kan vara ett numeriskt redskap för datakomprimering. Tidigare uttryckte vi $\mathbf{V}$ i sina kolumner $\mathbf{v}_i,\ i = 1, 2, \ldots, n$. Samma sak görs med matrisen $\mathbf{U}$ vars kolumner är $\mathbf{u}_i,\ i = 1, 2, \ldots, m$.

Vi ska visa att varje matris $\mathbf{A}$ kan skrivas som en linjärkombination av matriser $\mathbf{u}_i\mathbf{v}_i^T$. Här gör vi detta för fallet $m > n$:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m) \begin{pmatrix} s_1 & 0 & \cdots & 0 \\ 0 & s_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & s_n \\ 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{pmatrix} =$$

$$= (s_1\mathbf{u}_1, s_2\mathbf{u}_2, \ldots, s_n\mathbf{u}_n) \begin{pmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{pmatrix} = s_1\mathbf{u}_1\mathbf{v}_1^T + s_2\mathbf{u}_2\mathbf{v}_2^T + \cdots + s_n\mathbf{u}_n\mathbf{v}_n^T$$

Om $m \leq n$ kommer sista termen att ha index m. (Visa det!)

För att komprimera data sätter vi små singulärvärden till noll, det vill säga summan trunkeras efter lämpligt antal termer. En bildmatris med 500

rader och 400 kolumner innehåller tvåhundratusen element. Ifall tio singulärvärden räcker för att approximationen av bilden ska bli acceptabel, lagras endast $10\,(500 + 400) + 10 = 9010$ tal.

I sista kapitlet återkommer vi till en annan algoritm för bildkomprimering, nämligen tvådimensionell fouriertransform (FFT2D). Där finns bilder som visar komprimering dels med SVD dels med FFT2D.

Studera Heath avsnitt 3.6–3.9.

6.3 Numerisk lösning av Fredholms integralekvation

För många mätinstrument beskrivs apparatens uppförande av en integralekvation som går under beteckningen *Fredholms integralekvation av första slaget*: $\int_{-\infty}^{\infty} K(x,y)\,p(x)\,dx = f(y)$. Här är $p(x)$ en sökt funktion om vilken vi vet att den är noll utanför ett känt intervall $a < x < b$. Därför kan integrationen inskränkas att gälla detta intervall, och vi får

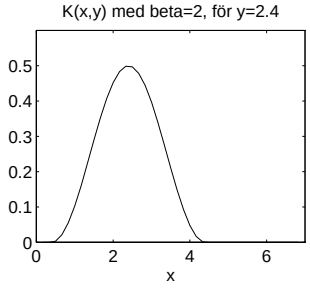
$$\int_a^b K(x,y)\,p(x)\,dx = f(y) \tag{4}$$

$K(x,y)$ är en känd apparatfunktion (K som i kärna eller engelskans *kernel*). Högerledet består i ett praktiskt fall av ett antal observerade värden för funktionen $f(y)$.

I det ideala fallet är $K(x,y)$ deltafunktionen, det innebär att den sökta parameterfunktionen p är identisk med utdatafunktionen f. Men idealfallet är mycket sällsynt, ett mer troligt utseende på apparatfunktionen $K(x,y)$ är följande, som vi ska utnyttja i ett simuleringsexempel:

$$K(x,y) = \frac{1}{2\beta}\left(1 + \cos\frac{\pi(y-x)}{\beta}\right), \quad |y-x| < \beta \quad \text{och} \quad K(x,y) = 0 \ \text{ annars.}$$

Om β-värdet är stort är mätinstrumentet inte särskilt bra, mätresultaten ger suddig och utsmetad information om den sökta parameterfunktionen.



Eftersom mätdata utgörs av m stycken observerade värden $f_i = f(y_i)$ kan vi skriva integralekvationen (4) på formen

$$\int_a^b K(x,y_i)\,p(x)\,dx = f_i, \quad i = 1,\,2,\,\ldots,m.$$

Vi har nu m ekvationer för att bestämma den okända funktionen $p(x)$. För att kunna göra en numerisk behandling av problemet delar vi x-intervallet i N delintervall av längden $\delta x = (b-a)/N$ och använder trapetsregeln för alla integralberäkningar. Integrandfunktionen för den i-te ekvationen är $F(x, y_i) = K(x, y_i)\, p(x)$. Trapetsregeln leder till integralapproximationen

$$I \approx \delta x \left[\frac{1}{2} F(a, y_i) + \sum_{j=1}^{N-1} F(a + j\,\delta x,\ y_i) + \frac{1}{2} F(b, y_i) \right]$$

som kräver fin indelning av x-intervallet, dvs stort N-värde för att noggrannheten ska bli bra.

Första och sista termen i trapetsregeln är noll i detta fall, eftersom det för parameterfunktionen antagits att $p(a) = 0$ och $p(b) = 0$. Med integralapproximationen insatt i (4) får vi

$$\delta x \sum_{j=1}^{n} K(x_j, y_i)\, p_j = f_i, \quad i = 1,\, 2,\, \ldots, m,$$

där x-värdena är ekvidistanta, $x_j = a + j\,\delta x$, och där $n = N - 1$ är antalet okända parametervärden $p_j = p(x_j)$.

Integralekvationen har alltså omformats till ett ekvationssystem $\mathbf{Ap} = \mathbf{f}$ med en $m \times n$-matris vars element A_{ij} består av kända apparatfunktionsvärden $K(x_j, y_i)$ multiplicerade med steget δx. Systemet kan vara överbestämt eller underbestämt beroende på antalet mätdata m och antalet intervall N i trapetsregeln. Ju fler intervall desto bättre integralapproximation, men det måste vägas mot att ekvationssystemet blir stort och alltmer underbestämt. Matrisen $\mathbf{A}$ med elementen $A_{ij} = K(x_j, y_i)\,\delta x$ är en bandmatris där bandbredden bestäms av apparatfunktionens β-värde.

I de flesta tillämpningar ger mätinstrumentet (om inte β är mycket litet) så suddiga mätresultat att problemet att finna parametrarna p_j blir illakonditionerat.

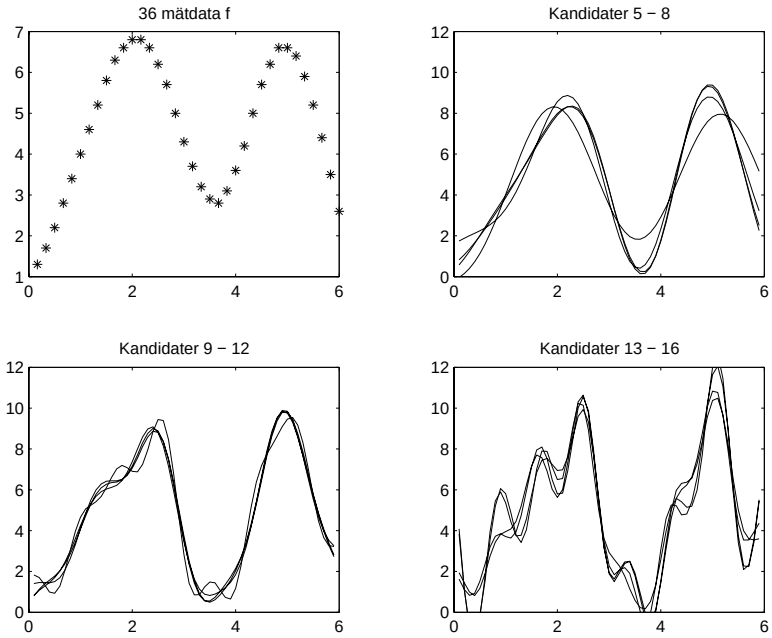
Algoritmen för Fredholms integralekvationsproblem går ut på att man singulärvärdesfaktoriserar matrisen och utnyttjar SVD för lösning av ekvationssystem. Med trunkerad SVD, sunt förnuft och eventuell insikt om karaktär hos parameterfunktionen försöker vi finna bästa tänkbara lösning. Vi betraktar de olika lösningskandidaterna och kan förkasta kandidater som oscillerar alltför kraftigt.

Vi tar ett simuleringsexempel med 36 mätpunkter i intervallet $[0,\, 6]$, erhållna med apparatfunktionen $K(x, y)$ med $\beta = 1.5$ applicerad på den för

oss gömda funktionen $p(x)$ (mätdata ses i första bilden nedan).

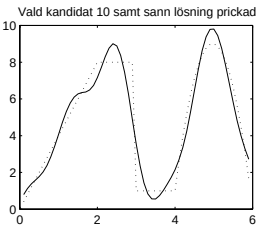
Vi vet att $p(0) = p(6) = 0$ och att $p(x) \geq 0$ i intervallet $0 \leq x \leq 6$. I trapetsregeln utnyttjas 60 intervall, vilket innebär att vi får en 36×59-matris. Antalet singulärvärden är 36, varav de sista tjugo är små (mindre än två promille av största singulärvärdet).

I bilderna två till fyra nedan visas resultatet, de så kallade lösningskandidaterna till $p(x)$, vid olika grad av trunkerad SVD. Av de sexton kandidaterna ska vi välja en bästa lösning. Sista bildens kurvor är alla så svängiga att vi förkastar dem. Någon av kandidaterna 9 – 12 är nog den bästa approximationen till $p(x)$ som vi kan få; vi bestämmer oss för kandidat nummer tio. I fantomfall som detta har vi tillgång till den sanna lösningen. På nästa sida visas den tillsammans med vår utvalda kandidat.



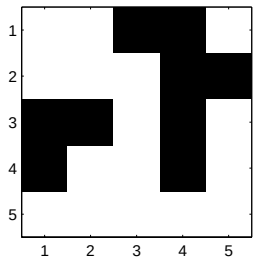

```
clear, clf
m=36;  dy=6/m; y=(dy:dy:6)';                    % m = antal data
N=60; dx=6/N; n=N-1; x=dx*(1:n); X=x';          % n = antal obekanta
beta=1.5
K=zeros(m,n);
for i=1:m
  for j=1:n, d=y(i)-x(j);
  if abs(d)<beta, K(i,j)=(1+cos(pi*d/beta))/(2*beta); end, end
end
```

```
  load f36
  subplot(2,2,1), plot(y,f36,'*'), title(['36 mätdata f']), hold on
  A=K*dx;
  [U,Sigma,V]=svd(A);  sigma=diag(Sigma)
  nsigma=length(find(sigma>0.002*sigma(1)))      % kasta små sing.värden
  d=U'*f36; z=d(1:nsigma)./sigma(1:nsigma);

% Titta på kandidatlösningarna 5 - 8,  9 - 12,  13 - 16
  pkand=V(:,1:4)*z(1:4);
  for b=2:4
    subplot(2,2,b)
    for k=(b-1)*4+1 :b*4
      pkand=pkand+V(:,k)*z(k); plot(X,pkand), axis([0 6 0 12]), hold on
    end
    title(['Kandidater ' int2str((b-1)*4+1) ' - ' int2str(b*4)])
  end
  nk=input('Vilken kandidat verkar bäst? ');
  pkoll=V(:,1:nk)*z(1:nk);
  subplot(2,2,1), plot(X,pkoll, X,ffred0(X),':')
  xlabel(['Vald kandidat ' int2str(nk) ' samt sann lösning prickad'])
```



I figuren är den valda tionde lösningskandidaten till parameterfunktionen den heldragna kurvan, medan den sanna fantomfunktionen $p(x)$ är den prickade kurvan.

6.4 Liten datortomografi

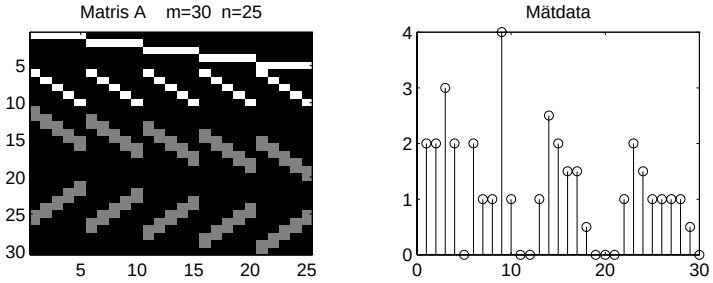


Vi ska studera ett plant datortomografiproblem som inte är större än att det är överblickbart. Inom en kvadrat finns en eller flera kroppar gömda och det gäller att lokalisera dem med hjälp av en mätapparat som lyser genom området i ett antal strålriktningar. Apparaten registrerar graden av svärtning, beroende på hur mycket kropp som finns i vägen för strålen.

De m erhållna mätvärdena lagras i vektorn $\mathbf{g}$. Om vårt rutnät är 5×5 finns $n = 25$ obekanta p-värden (så kallade pixels) för kroppslokaliseringen. Det leder till ett linjärt ekvationssystem $\mathbf{Ap} = \mathbf{g}$ där $m \times n$-matrisen $\mathbf{A}$ ges av mätriktningarna och pixelnumreringen.

Fem horisontella strålmätningar följt av fem vertikala ger de första tio raderna i matrisen $\mathbf{A}$ (förvissa dig om matrisstrukturen så långt). Därefter

följer tio strålar med 45^o positiv lutning — den elfte strålen som ger upphov till elfte matrisraden lyser bara genom rutnätets övre vänstra hörn (pixel nr 1) och tjugonde strålen genom rutnätets nedre högra hörn (pixel nr 25). Till slut görs tio mätningar med strålar med 45^o negativ lutning, rutnätets övre högra hörn (pixel nr 6) skannas av tjugoförsta strålen och nedre vänstra hörnet (pixel nr 21) skannas av trettionde strålen. I detta fall gäller alltså $m = 30$. De erhållna svärtningsvärdena för de trettio strålarna visas i högra figuren.



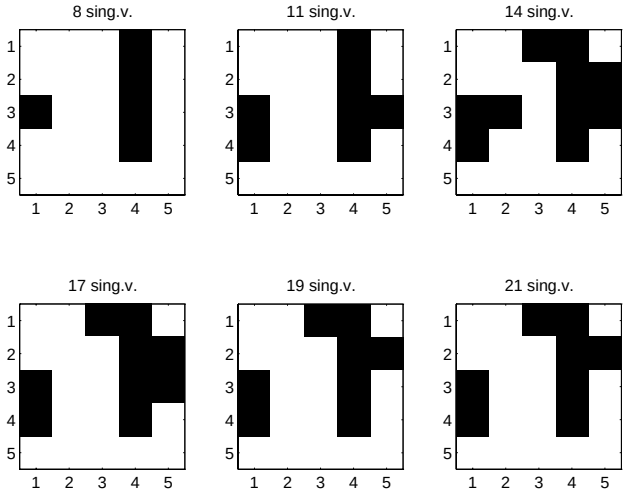
Minstakvadratlösningen till ekvationssystemet erhålls bäst med singulärvärdesfaktorisering. I vårt fall har vi flera mätningar än antal obekanta, men det visar sig när man tittar på singulärvärdena att endast 21 av dem är skilda från noll. Det innebär att alla mätningar faktiskt inte är oberoende av varandra.

Vid större problem med ett finare rutnät blir antalet obekanta mycket större, medan antalet mätvärden inte kan ökas så mycket. Många mätningar kan vara dyrbart (och obehagligt för den tomograferade patienten). Då får vi ett underbestämt ekvationssystem, men SVD fungerar ju även på sådana. Hur många singulärvärden som ska tas med får man pröva sig fram till.

Vår fantom antas ha värdet ett överallt där kropp finns och noll för övrigt. Vid rekonstruktionen, alltså efter lösning av ekvationssystemet, blir ju pixelvärdena mer utsmetade mellan noll och drygt ett. Vi avrundar alla värden som är mindre än 0.5 till 0, och alla värden större än 0.5 till 1 före uppritningen av de olika lösningskandidaterna.

```
% tomografi, liten datortomografi
  clear, figure(2), clf, figure(1), clf
  B=0.5*[1 0 0 0 0; 1 1 0 0 0; 0 1 1 0 0; 0 0 1 1 0; 0 0 0 1 1; 0 0 0 0 1];
  nb=5;  n=nb^2; m=30;           % n är antal obekanta, m antal mätdata
  A=zeros(m,n);                  % bygg upp matrisen A
  for i=1:5
    k5=5*(i-1)+1 : 5*i;
    A(i,k5)=ones(1,5);           % 5 horisontella mätningar
    A(6:10,k5)=eye(5);           % 5 vertikala mätningar
```

```
    A(10+i:15+i,k5)=B;           % 10 mätningar snett uppåt
    A(20+i:25+i,k5)=flipud(B);   % 10 mätningar snett nedåt
  end
  colormap(gray(3)),  subplot(2,2,1), imagesc(A)
  [U,S,V]=svd(A);  s=diag(S);  ns=length(find(s>0.001*s(1)))

% Mätdata, intensitetsdata
  ghv=[2 2 3 2 0  2 1 1 4 1]';             % data, strålar horis. och vertik.
  gu45=[0 0 1 2.5 2 1.5 1.5 0.5 0 0]';     % data, strålar snett uppåt
  gn45=[0 1 2 1.5 1 1 1 1 0.5 0]';         % data, strålar snett nedåt
  g=[ghv; gu45; gn45];                     % hela mätdatavektorn
  subplot(2,2,2), stem(1:m,g), title('Mätdata')
  utg=U'*g;
% Lösningskandidater beräknas och ritas upp
  figure(2),  colormap(gray(2))
  koll=[ns-13 ns-10 ns-7 ns-4 ns-2 ns];
  k=koll(1)-1;  z=utg(1:k)./s(1:k);  pkand=V(:,1:k)*z;  nr=1;
  for k=koll(1):ns
    zk=utg(k)/s(k);  pkand=pkand+V(:,k)*zk;
    if k==koll(nr)
      Q=pkand>=0 & pkand<=1; W=pkand>1; pp=Q.*pkand+W;  % fixar 0<=pp<=1
      Pcand=reshape(pp,nb,nb)';                        % matris av vektorn pp
      subplot(2,3,nr); imagesc(1-Pcand), axis image
      title([int2str(k) ' sing.v.']), drawnow, nr=nr+1;
    end
  end
```



7 Randvärdesproblem vid ODE

I grundkursen i numeriska metoder orienterades om inskjutningsmetoden och finitadifferensmetoden för hantering av randvärdesproblem vid ordinära differentialekvationer (ODE) av andra ordningen: $d^2y/dx^2 = f(x, y, dy/dx)$, med givna randvillkor $y(a) = y_0$ och $y(b) = y_{slut}$. Högerledet $f(x, y, y')$ utgörs av ett känt funktionsuttryck och sökt är $y(x)$ — eller snarare en god numerisk approximation till $y(x)$ — i intervallet $a \leq x \leq b$.

7.1 Inskjutningsmetoden

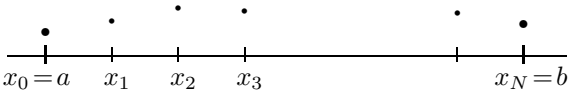
Inskjutningsmetoden innebär att randvärdesproblemet omformas till ett begynnelsevärdesproblem genom att man så bra som möjligt gissar begynnelsederivatan – alltså gör ett provskott. Så löser man differentialekvationen med bästa tänkbara metod, t ex Runge-Kuttas metod RK4 eller `ode45`, ser vad man får för slutvärde vid $x = b$; gissar nytt värde på startderivatan, löser differentialekvationen, ser på slutvärdet etc. Sekantmetoden[6] används för effektiv bestämning av startderivatavärdet, så att avvikelsen mellan det erhållna slutvärdet och det önskade slutvärdet y_{slut} ligger inom en i förväg bestämd felgräns.

7.2 Finitadifferensmetoden — FDM

Finitadifferensmetoden innebär ett helt annat angreppssätt av randvärdesproblemet. Vi betraktar till att börja med samma typ av randvärdesproblem som ovan, alltså en andra ordningens ODE med givna randvärden.

Gör en diskretisering av intervallet $a \leq x \leq b$, dvs dela intervallet i N delintervall med steget $h = (b - a)/N$.

Låt x_i beteckna de kända x-värdena $x_i = a + i\,h$ och y_i de okända y-värdena $y(x_i)$.



Antalet obekanta är $n = N - 1$ i typfallet med kända randvärden för $y(a)$ och $y(b)$. Nästa skede i algoritmen är att ersätta alla derivator med differenskvoter. Det innebär att differentialekvationen kommer att approximeras av en uppsättning differensekvationer. Approximationen blir bättre ju finare steg h som använts i diskretiseringen. Antalet differensekvationer är lika stort som antalet obekanta och tillsammans bildar de ett ekvationssystem för bestämning av de okända y_i-värdena.

[6] Se NAM avsnitt 6.4 eller Heath avsnitt 5.2.4.

Om differentialekvationen är linjär i y och dess derivator, alltså om den har formen $d^2y/dx^2 + p(x)\,dy/dx + q(x)\,y = f(x)$, leder finitadifferensmetoden till ett linjärt ekvationssystem med en tridiagonal systemmatris.

7.3 Derivataapproximationer

Differenskvotformler som approximation till förstaderivatan studeras i NAM avsnitt 1.3. Där visas hur man med hjälp av taylorutveckling kan härleda trunkeringsfelet. För framåtdifferenskvoten $y'(x_i) \approx (y_{i+1} - y_i)/h$ är trunkeringsfelet proportionellt mot steget h, vilket brukar skrivas $O(h)$. För centraldifferenskvoten $y'(x_i) \approx (y_{i+1} - y_{i-1})/2h$ är felet $O(h^2)$.

Vid behandling av randvärdesproblem med FDM vill vi ha goda derivataapproximationer och utnyttjar därför centraldifferenskvoten för förstaderivatan. För andra-, tredje- och fjärdederivatorna kan nedanstående differensformler härledas, samtliga med trunkeringsfel $O(h^2)$.

$$\begin{aligned} y'(x_i) &\approx \tfrac{1}{2h}(y_{i+1} - y_{i-1}) \\ y''(x_i) &\approx \tfrac{1}{h^2}(y_{i+1} - 2y_i + y_{i-1}) \\ y'''(x_i) &\approx \tfrac{1}{2h^3}(y_{i+2} - 2y_{i+1} + 2y_{i-1} - y_{i-2}) \\ y''''(x_i) &\approx \tfrac{1}{h^4}(y_{i+2} - 4y_{i+1} + 6y_i - 4y_{i-1} + y_{i-2}) \end{aligned} \tag{5}$$

Studera Heath avsnitt 10.1, 10.2, 10.3, 10.4.

7.4 Två exempel, FDM på linjärt och ickelinjärt problem

I NAM avsnitt 8.7.3 finns en detaljerad behandling av finitadifferensmetoden på följande linjära randvärdesproblem:

$$y'' + a\sqrt{x}\,y' = (6 - y)x, \;\; y(0) = 1, \;\; y(4) = -1, \;\; 0 \le x \le 4,$$

för några olika värden på parametern a. Studera algoritmen!

Randvärdesproblemets differentialekvation kan vara mer komplicerad med ickelinjära y- och y'-förekomster. Finitadifferensmetoden leder i dessa fall till ett ickelinjärt ekvationssystem för bestämning av y_i-värdena. Det löses iterativt med Newtons metod som kräver goda startgissningar, vilket kan kräva visst trixande i beräkningsarbetet. Jacobianmatrisen blir i FDM-tillämpningar alltid en bandmatris.

Den ickelinjära differentialekvationen $y'' + c\,y^2 = (6-y)x$ med parametervärdet $c = 0.1$ och med randvärdena $y(0) = 1, \quad y(4) = -1$ behandlas utförligt i NAM avsnitt 8.7.4. Studera lösningsförfarandet med det taktiska greppet att först angripa det linjära problemet då $c = 0$ och sedan etappvis göra problemet alltmer ickelinjärt. Detta förfarande som innebär att man uppdaterar startgissningarna successivt vid tilltagande ickelinjaritet kallas på engelska "imbedding" (försvenskas till *inbäddning*).

7.5 Olika typer av randvillkor

I exemplen hittills har randvillkoren varit av den snälla typen som kallas *dirichletvillkor* och som innebär att funktionsvärdet är känt i randpunkten.

Vid FDM diskretiseras intervallet $a \le x \le b$ i N stycken delintervall med $h = (b-a)/N$ enligt figuren tidigare. När båda randvillkoren är av dirichlettyp är $y(a) = y_0$ och $y(b) = y_N$ kända och antalet obekanta blir $n = N - 1$. En specialbehandling av differensekvationerna måste göras för $i = 1$ (som innehåller det kända y_0 samt de obekanta y_1, y_2) och för $i = n$ (som innehåller y_{n-1}, y_n och det kända y_N).

Om ett randvillkor innehåller en derivata, t ex på vanligaste formen $y'(b) = 0$ eller blandformen $y'(b) - 0.1(y(b) - 17) = 0$, kallas villkoret för *neumannvillkor*. Då blir situationen något knepigare. Dels måste den ingående derivatan approximeras med en differenskvot av samma noggrannhetsordning som övriga derivataapproximationer. Dels måste man noga tänka över hur antalet obekanta n hänger ihop med antalet diskretiseringsintervall N. I själva verket är detta den egentliga stötestenen vid FDM och kräver alltid stor omsorg!

Anta att vi har ett randvärdesproblem med känt $y(a) = y_0$, men med högra randvillkoret $y'(b) - 0.1(y(b) - 17) = 0$. Eftersom värdet $y(b) = y_N$ inte är känt, kommer antalet obekanta att vara lika många som antalet intervall, alltså $n = N$.

Approximera derivatan vid $x_n = b$ med centraldifferenskvot. För att kunna göra det införs en tillfällig spökpunkt y_{n+1}. Neumannvillkoret ersätts av $(y_{n+1} - y_{n-1})/2h - 0.1(y_n - 17) = 0$, som ger $y_{n+1} = y_{n-1} + 0.2h(y_n - 17)$.

Specialbehandling måste även nu göras av första och sista differensekvationen, alltså för $i = 1$ och för $i = n = N$. I den sista ekvationen ingår y_{n-1}, y_n och y_{n+1}, men spökpunkten y_{n+1} är inte önskvärd utan ersätts av $y_{n-1} + 0.2h(y_n - 17)$. Sista differensekvationen kommer därför att kunna uttryckas i enbart y_{n-1} och y_n.

7.6 Utböjning av en cirkelplatta

En cirkelplatta med radien $R = 60$ mm och tjockleken $\tau = 2.0$ mm är längs periferin fritt upplagd på en cirkelformad ram. Cirkelplattan belastas med ett jämnt fördelat radiellt beroende tryck $q(r)$. Utböjningen kommer därför enbart att få radiellt beroende och vi vill beräkna utböjningen $u(r)$ för $0 \le r \le R$. Den bestäms av följande fjärde ordningens differentialekvation där E, γ och $\nu_{\varphi r}$ är givna materialkonstanter:

$$\frac{d^4u}{dr^4} + \frac{2}{r}\frac{d^3u}{dr^3} - \frac{\gamma}{r^2}\frac{d^2u}{dr^2} + \frac{\gamma}{r^3}\frac{du}{dr} = -\frac{q(r)}{D}, \quad \text{där} \quad D = \frac{\tau^3 E}{12} \tag{6}$$

Vid randen $r = R$ gäller att utböjningen och momentet M_r båda är noll. Eftersom momentet bestäms av $M_r(r) = -D\,(u'' + \nu_{\varphi r} u'/r)$ får vi

$$u(R) = 0 \;\text{ och }\; u''(R) + \nu_{\varphi r} u'(R)/R = 0. \tag{7}$$

Vid $r = 0$ bestäms randvillkoren av symmetrin, alla derivator av udda ordning måste vara noll: $u'(0) = 0$ och $u'''(0) = 0$ (utböjningen är en jämn funktion av variabeln r).

Vi tillämpar FDM och börjar med att diskretisera intervallet $0 \le r \le R$ i N delintervall: $r_i = i \cdot h$ med $h = R/N$. Approximera alla ingående derivator i (6) med differenskvoterna enligt formlerna (5). Efter multiplicering av båda sidor med h^4 uppkommer differensekvationerna

$$(u_{i+2} - 4u_{i+1} + 6u_i - 4u_{i-1} + u_{i-2}) + \frac{2h}{2r_i}\,(u_{i+2} - 2u_{i+1} + 2u_{i-1} - u_{i-2}) -$$
$$-\frac{\gamma h^2}{r_i^2}\,(u_{i+1} - 2u_i + u_{i-1}) + \frac{\gamma h^3}{2r_i^3}\,(u_{i+1} - u_{i-1}) = -\frac{h^4 q(r_i)}{D}$$

Att multiplicera båda sidor med det inverterade högerledsuttrycket $\frac{D}{h^4 q(r_i)}$ är ett trick som inte är nödvändigt här men som underlättar vid egensvängningsberäkningarna senare.

Efter sortering av termerna ovan och multiplikation med $m_i = \frac{D}{h^4 q(r_i)}$ erhålls

$$a_i u_{i-2} + b_i u_{i-1} + c_i u_i + d_i u_{i+1} + e_i u_{i+2} = f_i \quad \text{med} \tag{8}$$

$$\begin{cases} a_i = m_i(1 - \frac{h}{r_i}) \\ b_i = m_i(-4 + \frac{2h}{r_i} - \frac{\gamma h^2}{r_i^2} - \frac{\gamma h^3}{2r_i^3}) \\ c_i = m_i(6 + \frac{2\gamma h^2}{r_i^2}) \\ d_i = m_i(-4 - \frac{2h}{r_i} - \frac{\gamma h^2}{r_i^2} + \frac{\gamma h^3}{2r_i^3}) \\ e_i = m_i(1 + \frac{h}{r_i}) \\ f_i = -1 \end{cases}$$

Kring cirkelplattans mittpunkt $r=0$ råder symmetri. Utböjningsfunktionen $u(r)$ blir då en jämn funktion med egenskapen att alla uddaderivator är noll.

$$\text{Villkoret } \; u'(0)=0 \; \text{ approximeras av } \; \frac{u_1-u_{-1}}{2h}=0, \;\; \text{dvs } \; u_{-1}=u_1. \quad (9)$$

Utböjningen vid $r=r_0=0$ utgörs av det okända värdet u_0. Hur får vi in det i den numeriska behandlingen? En differensekvation för $i=0$ är ju inte lämpligt att ställa upp, eftersom r_0 finns i nämnaren på många ställen.

Det finns dock ett numeriskt knep. Vi vet att $u(r)$ är en jämn funktion. Serieutveckla kring $r=0$. Endast jämna potenser av r ingår, och för tillräckligt litet r kan man bryta efter r^2-termen, alltså $u(r)=u_0+C\,r^2+O(r^4)$.
Sätt först in $r=h$ i formeln, det leder till $u(h)=u_1\approx u_0+C\,h^2$.
Sätt sedan in $r=2h$ i formeln, det ger $\;u(2h)=u_2\approx u_0+C\,(2h)^2$.

Eliminera C genom att multiplicera första ekvationen med 4 och därefter subtrahera den andra ekvationen. Det ger $4u_1-u_2=3u_0$. Värdet u_0 kan alltså uttryckas i u_1 och u_2 (med ett fel $O(h^4)$) och blir

$$u_0=(4u_1-u_2)/3. \quad (10)$$

Randvillkoren (9) och (10) kan sättas in i differensekvationen (8) för $i=1$ och $i=2$:

$$\begin{aligned}
i=1: \quad & a_1u_{-1}+b_1u_0+c_1u_1+d_1u_2+e_1u_3=f_1\\
\Longrightarrow \quad & a_1u_1+b_1(4u_1-u_2)/3+c_1u_1+d_1u_2+e_1u_3=f_1\\
\Longrightarrow \quad & (c_1+a_1+\tfrac{4}{3}b_1)u_1+(d_1-\tfrac{1}{3}b_1)u_2+e_1u_3=f_1\\
i=2: \quad & a_2u_0+b_2u_1+c_2u_2+d_2u_3+e_2u_4=f_2\\
\Longrightarrow \quad & a_2(4u_1-u_2)/3+b_2u_1+c_2u_2+d_2u_3+e_2u_4=f_2\\
\Longrightarrow \quad & (b_2+\tfrac{4}{3}a_2)u_1+(c_2-\tfrac{1}{3}a_2)u_2+d_2u_3+e_2u_4=f_2
\end{aligned}$$

Nu tar vi itu med randen vid $r=R$. Eftersom vi vet att $u(R)=0$, så är det värdet vid $r=R-h$ som kommer sist av de n obekanta u-värdena och därmed betecknas u_n. Antalet obekanta är alltså $n=N-1$. Derivatorna i randvillkoret (7) approximeras av differenskvoter, vilket kräver tillgång till två spökpunkter u_{n+1} och u_{n+2}. De båda randvillkoren i (7) kan då uttryckas:

$$u_{n+1}=0 \;\text{ och }\; \frac{u_{n+2}-2u_{n+1}+u_n}{h^2}+\frac{\nu_{\varphi r}}{R}\cdot\frac{u_{n+2}-u_n}{2h}=0.$$

Lös ut u_{n+2} och utnyttja att $u_{n+1}=0$. Det leder till följande samband:

$$u_{n+1}=0 \;\text{ och }\; u_{n+2}=\kappa\,u_n \;\text{ där }\; \kappa=(\frac{\nu_{\varphi r}h}{2R}-1)/(\frac{\nu_{\varphi r}h}{2R}+1)$$

Dessa uttryck sätts in i differensekvationen (8) för $i = n-1$ och $i = n$:

$$i = n-1: \; a_{n-1}u_{n-3} + b_{n-1}u_{n-2} + c_{n-1}u_{n-1} + d_{n-1}u_n + e_{n-1}u_{n+1} = f_{n-1}$$
$$\Longrightarrow \; a_{n-1}u_{n-3} + b_{n-1}u_{n-2} + c_{n-1}u_{n-1} + d_{n-1}u_n = f_{n-1}$$

$$i = n: \; a_n u_{n-2} + b_n u_{n-1} + c_n u_n + d_n u_{n+1} + e_n u_{n+2} = f_n$$
$$\Longrightarrow \; a_n u_{n-2} + b_n u_{n-1} + (c_n + \kappa e_n)u_n = f_n$$

De övriga differensekvationerna (från $i = 3$ till och med $i = n-2$) påverkas inte av randvillkoren.

Nu kan vi sammanställa allt i ett ekvationssystem på matrisform $\mathbf{Au} = \mathbf{f}$ där matrisen får följande utseende:

$$\mathbf{A} = \begin{pmatrix} c_1+a_1+\frac{4}{3}b_1 & d_1-\frac{1}{3}b_1 & e_1 & 0 & 0 & 0 & ... & 0 \\ b_2+\frac{4}{3}a_2 & c_2-\frac{1}{3}a_2 & d_2 & e_2 & 0 & 0 & ... & 0 \\ a_3 & b_3 & c_3 & d_3 & e_3 & 0 & \dots & 0 \\ 0 & a_4 & b_4 & c_4 & d_4 & e_4 & \dots & 0 \\ 0 & 0 & . & . & . & . & . & 0 \\ . & \dots & . & . & . & . & . & . \\ . & \dots & . & . & . & . & . & . \\ 0 & \dots & 0 & a_{n-2} & b_{n-2} & c_{n-2} & d_{n-2} & e_{n-2} \\ 0 & 0 & \dots & 0 & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ 0 & 0 & 0 & \dots & 0 & a_n & b_n & c_n + \kappa e_n \end{pmatrix} \tag{11}$$

Det är en bandmatris $\mathbf{A}$ med bandbredden fem. Högerledet $\mathbf{f}$ är en vektor med komponenterna $f_i = -1$. Lösningen av det linjära systemet utgör de obekanta utböjningsvärdena u_i, $i = 1, 2, ..., n$. När de är kända får vi också ut nedböjningen vid $r = 0$ ur sambandet (10).

Matlabs sparse-matrisfinesser

Vi ska pröva på att utnyttja MATLABs sparsematrisfinesser som gör beräkningar med glesa matriser mycket effektivare än den vanliga fullsystemlösaren. Lagringen av bandmatrisen görs minnessnålt — nollorna utanför bandet är ointressanta. Så här definieras en $n \times n$ bandmatris med fem diagonaler:

```
A=spdiags(Aband,-2:2,n,n);
```

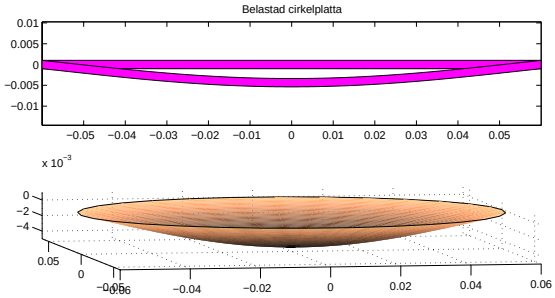
Den vanliga satsen `u=A\f` används på den vällagrade matrisen, men nu löses systemet på effektivt sätt (motsvarar ungefär kurskatalogens `tridia` för tridiagonala system).

I vårt fall ska `Aband` byggas upp så här:

$$\begin{pmatrix} a_3 & b_2+\frac{4}{3}a_2 & c_1+a_1+\frac{4}{3}b_1 & 0 & 0 \\ a_4 & b_3 & c_2-\frac{1}{3}a_2 & d_1-\frac{1}{3}b_1 & 0 \\ a_5 & b_4 & c_3 & d_2 & e_1 \\ . & . & . & . & . \\ . & . & . & . & . \\ a_n & b_{n-1} & c_{n-2} & d_{n-3} & e_{n-4} \\ 0 & b_n & c_{n-1} & d_{n-2} & e_{n-3} \\ 0 & 0 & c_n+\kappa e_n & d_{n-1} & e_{n-2} \end{pmatrix}$$

Nu kan vi skriva ett program för beräkning och uppritning av utböjningen för cirkelplattan med $R = 0.060$ och $\tau = 0.002$ (längdenhet m). Materialkonstanterna är $E = 1.0 \cdot 10^8$ (styvhet i N/m^2), $\gamma = 3.6$ och $\nu_{\varphi r} = 0.24$ (dimensionslösa). Trycket på plattan är $q(r) = 300\,(1 - r^2/R^2)$ (N/m^2).

Låt oss börja med att dela intervallet $0 \le r \le R$ i $N = 25$ delintervall, alltså 24 obekanta. Vi upprepar sedan beräkningen med 50, 100, 200 och 400 intervall — ju finare indelning, desto bättre noggrannhet.



För utböjningen u_0 i mitten av cirkelplattan erhålls värdena (här i mm): -4.191 -4.289, -4.336, -4.359, -4.369. Det krävs alltså 400 intervall för att resultatet ska få mer än två korrekta siffror.

```
% cirkel1platta
  clear, clf
  E=100e6; g=3.6; v=0.24;     % materialkonstanter
  tau=0.002;                  % 2 mm tjock platta
  R=0.060;                    % radie 6 cm
  q0=300;
  D=E*tau^3/12;
  N=25; unoll=[];
  for nr=1:5                  % Lös med N=25,50,100,200,400
    n=N-1; h=R/N;  r=h*(1:n)';
    q=q0*(1-r.^2/R^2);
    m=D./(q*h^4);
    alfa=h*v/(2*R); k=(alfa-1)/(alfa+1);
    a=m.*(1-h./r);
    b=m.*(-4+2*h./r-g*h^2./r.^2-g*h^3./(2*r.^3));
    c=m.*(6+2*g*h^2./r.^2);
```

```
  d=m.*(-4-2*h./r-g*h^2./r.^2+g*h^3./(2*r.^3));
  e=m.*(1+h./r);
  f=-ones(n,1);

  dia=c;
  dia(1)=c(1)+a(1)+4/3*b(1); dia(2)=c(2)-a(2)/3; dia(n)=c(n)+k*e(n);
  sup1=d(1:n-1);  sup1(1)=d(1)-b(1)/3;   sup2=e(1:n-2);
  sub1=b(2:n);    sub1(1)=b(2)+4/3*a(2); sub2=a(3:n);

% Lös med Matlabs sparse
  Aband=[[sub2;0;0] [sub1;0] dia [0;sup1] [0;0;sup2]];
  A=spdiags(Aband,-2:2,n,n);
  u=A\f;
  u0=(4*u(1)-u(2))/3; unoll=[unoll u0];       % värdet vid r=0
  rr=[ 0; r; R];
  uu=[u0; u; 0];

% 3D-bild av utböjningen
  if nr==2
    fi=0:2*pi/30:2*pi; cf=cos(fi); sf=sin(fi);
    X=rr*cf; Y=rr*sf; Z=uu*ones(size(fi));
    surf(X,Y,Z), hold on, colormap copper, shading flat
    plot3(R*cf,R*sf,0*fi), axis([-R R -R R -0.005 0.001])
  end
  N=2*N;
end
u0_mm=1000*unoll         % nedböjning i mm
```

7.7 Egenvärdesproblem vid ordinära differentialekvationer

Vi ska nu studera egenvärdesproblemet för differentialekvationen

$$G(x)\frac{d^2y}{dx^2} = \alpha\, y, \quad y(0) = 0, \quad y(L) = 0,$$

där $G(x)$ är en känd funktion. Man önskar bestämma det till beloppet minsta värde α, för vilket en icketrivial lösning $y(x)$ existerar för $0 \le x \le L$.

För numerisk behandling använder vi finitadifferensmetoden och kan därigenom överföra differentialekvationsproblemet till ett egenvärdesproblem för en matris. Dela intervallet i N delar (t ex $N = 50$) med längden $h = L/N$ och diskretiseringspunkter vid $x_i = i \cdot h,\ i = 0,\ 1,\ 2, ..., N$.
Randvärdena y_0 och y_N är kända, medan $y_1,\ y_2, ...,\ y_{N-1}$ är obekanta komponenter; vi inför $n = N - 1$.

Differentialekvationen approximeras nu av differensekvationerna

$$G(x_i)\,\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = \alpha\, y_i, \quad i = 1,\ 2, \ldots, n.$$

Vi inför beteckningen $g_i = G(x_i)/h^2$ och sorterar om termerna:

$$g_i y_{i-1} - 2g_i y_i + g_i y_{i+1} = \alpha\, y_i, \;\; i = 1,\, 2, \ldots, n.$$

Granska första och sista ekvationen som innehåller randvärdena. Eftersom dessa är noll får vi: $-2g_1 y_1 + g_1 y_2 = \alpha y_1$ och $g_n y_{n-1} - 2g_n y_n = \alpha y_n$.
Allt kan nu sammanfattas i matrisform:

$$\begin{pmatrix} -2g_1 & g_1 & 0 & 0 & \cdots & 0 \\ g_2 & -2g_2 & g_2 & 0 & \cdots & 0 \\ 0 & g_3 & -2g_3 & g_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & g_{n-1} & -2g_{n-1} & g_{n-1} \\ 0 & 0 & 0 & 0 & g_n & -2g_n \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} = \alpha \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix}$$

Problemet har nu omformats till ett egenvärdesproblem $\mathbf{Ay} = \alpha\mathbf{y}$ för en tridiagonal $n \times n$ matris. Inversa potensmetoden är lämplig för beräkning av det till beloppet minsta egenvärdet till det approximerande matrisproblemet.

För att få en uppfattning om tillförlitligheten bör beräkningarna göras om med halverat steg h i diskretiseringen (fördubbling av antalet intervall N). Upprepa tills det beräknade α-värdet överensstämmer med föregående med önskad tolerans. Eftersom differenskvoten som utnyttjas som approximation till andraderivatan har trunkeringsfelet $O(h^2)$, kan noggrannheten förbättras genom richardsonextrapolation (se NAM avsnitt 3.8).

7.8 Egensvängningar för cirkelplattan

Vi betraktar samma cirkelplatta som i utböjningsproblemet (6), men nu gäller det att finna de lägsta egensvängningstillstånden hos plattan när den utsätts för trycket $q(r) = 300(1 - r^2/R^2)$. Problemet är ett egenvärdesproblem för differentialekvationen

$$\frac{d^4u}{dr^4} + \frac{2}{r}\frac{d^3u}{dr^3} - \frac{\gamma}{r^2}\frac{d^2u}{dr^2} + \frac{\gamma}{r^3}\frac{du}{dr} = \frac{q(r)}{D}\omega^2 u$$

med samma homogena randvillkor och samma materialdata som tidigare. Konstanten ω i högerledet är vinkelfrekvensen för den harmoniska svängningsrörelse i tiden som plattan utsätts för. Vissa frekvenser, de så kallade egenfrekvenserna, ger resonans i plattan. Vi vill bestämma de tre lägsta egenfrekvenserna och tillhörande egenfunktioner.

Utnyttja samma finitadifferensapproximation som vid cirkelplattans utböjning och samma multiplikationstrick med $m_i = \frac{D}{h^4 q(r_i)}$.

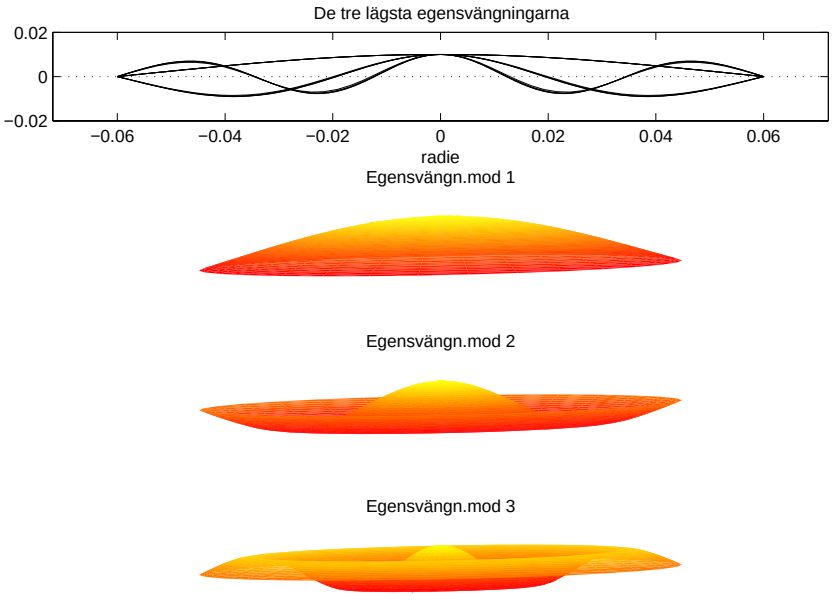
Det innebär att vänsterledet får formen $\mathbf{Au}$ med exakt samma matris (11) som i utböjningsproblemet. Högerledet blir $\omega^2\mathbf{u}$ vilket kan skrivas $\lambda\mathbf{u}$. (Multiplikationstricket innebär att inget r-beroende finns kvar i högerledet).

Nu har problemet omvandlats till ett egenvärdesproblem $\mathbf{Au} = \lambda\mathbf{u}$ för den glesa matrisen $\mathbf{A}$. När ett visst egenvärde λ och tillhörande egenvektor (egensvängningsmod) har räknats ut bestäms tillhörande egenfrekvens ω ur $\omega = \sqrt{\lambda}$.

För att få god noggrannhet kan en tät diskretisering behövas, vilket innebär att matrisen blir stor och `eig(A)` tidskrävande. Vi är ju endast intresserade av de tre till beloppet minsta egenvärdena med tillhörande egenvektorer. I MATLAB finns `[V,D]=eigs(A,antal,'sm')` som klarar av detta och som kommer att utnyttjas i koden.

En effektiv alternativ metod om man inte har tillgång till `eigs(A)` är inversa potensmetoden med skift. För att få lämpliga skift löser man egenvärdesproblemet med `eig(A)` för en grov diskretisering, förslagsvis 25 intervall då matrisen är 24×24. Med `sort(eig(A))` fås sorterade egenvärden. De tre erhållna minsta värdena får utgöra skiftvärdena i inversa potensmetoden för beräkning av önskade egenvärden och egenvektorer vid en tät diskretisering.

Figurerna nedan visar egensvängningsmoderna för grundfrekvensen och de två närmast högre egenfrekvenserna för vår cirkelplatta. Beräkningar har gjorts för diskretisering i 25, 50, 100 och 200 intervall. Egenfrekvenserna blir $\omega = 16.9,\ 147,\ 383$.



```
% cirkel1svang
% Egensvängningar för cirkelplattan
  ... ...
  % Identiskt med cirkel1platta
  ... ...
  D=E*tau^3/12;

  subplot(4,1,1), plot([-1.2*R 1.2*R],[0 0],':')
  axis([-1.2*R 1.2*R -0.02 0.02]), hold on
  title('De tre första egensvängningarna'), xlabel('radie')

  N=25;
  for nr=1:4                          % Lös med N=25, 50, 100, 200
    ... ...
  % Identiskt med cirkel1platta
    ... ...
    A=spdiags(...);

    rr=[0; r; R];
    opts.disp=0;                      % Undertrycker iterationsutskrift
    [V,Lamb]=eigs(A,3,'sm',opts);
    lamb=diag(Lamb)'
    for mod=1:3
      y=V(:,mod);                     % Egenvektor
      y0=(4*y(1)-y(2))/3;             % Värde vid r=0 enligt formel(10)
      yy=[y0; y; 0];
      ysvang=yy*sign(y0)/norm(yy,inf)*0.01; % Normera, maxvärde 0.01
      subplot(4,1,1)
      plot(rr,ysvang,-rr,ysvang)

    % 3D-bild av egensvängningarna vid diskretisering i 50 intervall
      if nr==2
        fi=0:2*pi/30:2*pi;
        X=rr*cos(fi); Y=rr*sin(fi); Z=ysvang*ones(size(fi));
        subplot(4,1,1+mod), surf(X,Y,Z)
        hold on, shading flat, axis off
        title(['Egensvängn.mod ' int2str(mod)])
      end
    end
    N=2*N;
  end
  omega=sqrt(lamb)
```

7.9 Galerkins metod för randvärdesproblem

Den idé som framgångsrikt utnyttjas vid minstakvadratproblem, alltså att modellera en sökt funktion $y(x)$ med en linjärkombination av basfunktioner

$$y(x) \approx \sum c_k \phi_k(x)$$

är tillämplig också vid behandling av randvärdesproblem vid differentialekvationer. Vi ska studera två lösningsförfaranden, *Galerkins metod* och *kollokation*, som båda grundar sig på ansatsen med basfunktioner och tillhör familjen finitaelementmetoder (FEM). Det innebär att man tillämpar ett helt annat angreppssätt än vid finitadifferensmetoden (FDM).

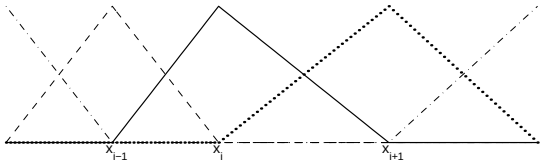
7.9.1 Ansats med hattfunktioner

Låt oss betrakta det linjära randvärdesproblemet:

$$y'' + p(x)\,y' + q(x)\,y = f(x), \quad a \le x \le b, \quad y(a) = \alpha, \;\; y(b) = \beta, \qquad (12)$$

där $p(x)$, $q(x)$ och $f(x)$ är kända funktioner. Det gäller att approximera lösningen till differentialekvationen med en ändlig linjärkombination $\sum c_k \phi_k(x)$ av på förhand valda basfunktioner. I Galerkins metod är det naturligast och enklast att låta basfunktionerna bestå av linjära splines, de så kallade hattfunktionerna $H_i(x)$ som presenterades i avsnitt 5.2.1. Nedan betecknar vi dock hattfunktionerna med $\phi_i(x)$ och numrerar dem från noll till N.

För knutpunktsplaceringen behövs en indelning $x_0, x_1, \ldots, x_N$ av området $a \le x \le b$, där x_0 och x_N sammanfaller med ändpunkterna, alltså $x_0 = a$ och $x_N = b$. Vi inför som vanligt $h_i = x_{i+1} - x_i$.



Basfunktionerna $\phi_i(x)$ är noll överallt utom i intervallet $[x_{i-1},\, x_{i+1}]$ där de ser ut som toppiga hattar och definieras av

$$\phi_i(x) = \begin{cases} (x - x_{i-1})/h_{i-1}, & x_{i-1} \le x \le x_i \\ (x_{i+1} - x)/h_i, & x_i \le x \le x_{i+1} \end{cases} \qquad (13)$$

De yttersta hattfunktionerna ϕ_0 och ϕ_N kan betraktas som halva hattar: $\phi_0(x) = (x_1 - x)/h_0,\; x_0 \le x \le x_1$ och noll för övrigt. För den högra gäller $\phi_N(x) = (x - x_{N-1})/h_{N-1},\; x_{N-1} \le x \le x_N$.

Med Galerkins metod söker man en god approximation $u(x) = \sum_{k=0}^{N} c_k \phi_k(x)$ till differentialekvationens exakta lösning $y(x)$.

När en funktion $u(x)$ skrivs som en linjärkombination av hattfunktioner gäller sambanden $u(x_i) = c_i,$ och $u(x)$ blir den polygonkurva som passerar genom punkterna $(x_i,\, c_i),\ i = 0, 1, 2, \ldots, N$ (se även avsnitt 5.2.1). Denna förträffliga egenskap innebär att koefficienten c_0 överensstämmer med randvärdet $y(a) = \alpha$ och att c_N överensstämmer med $y(b) = \beta,$ alltså

$$c_0 = \alpha, \quad c_N = \beta. \tag{14}$$

Kvar finns $n = N - 1$ stycken okända koefficienter $c_1, \ldots, c_n$, och vi måste kunna ställa upp n ekvationer för att bestämma dessa.

7.9.2 Galerkins ortogonalitetsvillkor

Liksom man vid minstakvadratmetoden utnyttjar den viktiga egenskapen att residualvektorn är ortogonal mot basvektorerna (NAM avsnitt 2.3), så finns motsvarande ortogonalitetsvillkor i Galerkins metod.

Vi behöver en definition för begreppet ortogonalitet hos funktioner:

> Två funktioner f och g är ortogonala på intervallet $[a,\, b]$ om
>
> $$\int_a^b f(x)g(x)\,dx = 0.$$
>
> Jämför motsvarande definition när det gäller vektorer:
> **f** och **g** är ortogonala om $\mathbf{f}^T\mathbf{g}$=0.

Bilda residualen $r(x)$ bestående av differentialekvationens högerled minus dess vänsterled, där $y(x)$ i (12) ersätts av sin approximerande funktion $u(x)$:

$$r(x) = f(x) - (u''(x) + p(x)u'(x) + q(x)u(x)). \tag{15}$$

Avsikten är nu att försöka bestämma koefficienterna i $u(x)$ så att residualfunktionen blir ortogonal mot basfunktionerna, dvs så att följande ortogonalitetsvillkor är uppfyllt:

$$\int_a^b r(x)\phi_i(x)\,dx = 0. \tag{16}$$

Insättning av residualfunktionen (15) (med ombytt tecken) leder till

$$\int_a^b (u'' + p\,u' + q\,u - f)\,\phi_i\,dx = 0, \quad i = 1, 2, \ldots, n \tag{17}$$

där u står för linjärkombinationen av hattfunktioner.

Första termen i (17) som innehåller u'' medför lite trassel på grund av den ickeexisterande andraderivatan hos hattfunktionerna. Men på u''-termen i integralen kan partiell integration utföras:

$$\int_a^b u''\phi_i\,dx = [u'(x)\phi_i(x)]_a^b - \int_a^b u'\phi_i'\,dx = -\int_a^b u'\phi_i'\,dx$$

Den utintegrerade delen $u'(b)\phi_i(b) - u'(a)\phi_i(a)$ är noll eftersom hattfunktionerna för $i = 1, 2, ..., n$ uppfyller $\phi_i(a) = \phi_i(b) = 0$.

Sambanden (17) kan därmed skrivas

$$-\int_a^b u'\phi_i'\,dx + \int_a^b \left(p\,u' + q\,u\right)\phi_i\,dx = \int_a^b f\,\phi_i\,dx$$

Vi stoppar in $u = \sum_{k=0}^N c_k\phi_k$ och $u' = \sum_{k=0}^N c_k\phi_k'$ och kastar om integrations- och summationsordningen:

$$\sum_{k=0}^N c_k\left(-\int_a^b \phi_k'\phi_i'dx + \int_a^b p\,\phi_k'\phi_i dx + \int_a^b q\,\phi_k\phi_i dx\right) = \int_a^b f\,\phi_i\,dx$$

Låt allt inom parentesen i vänsterledet betecknas a_{ik}:

$$a_{ik} = -\int_a^b \phi_k'\phi_i'dx + \int_a^b p\,\phi_k'\phi_i dx + \int_a^b q\,\phi_k\phi_i dx \tag{18}$$

och inför beteckningen f_i för högerledsintegralen. Nu kan den i-te ekvationen i kompakt form skrivas

$$\sum_{k=0}^N a_{ik}c_k = f_i. \tag{19}$$

Alla ekvationer, $i = 1, 2, \ldots, n$, samlas i ett linjärt ekvationssystem $\mathbf{Ac} = \mathbf{f}$.

7.9.3 Egenskaper hos systemmatrisen

När basfunktionerna utgörs av hattfunktioner kommer matrisen $\mathbf{A}$ att bli tridiagonal. Hur inser man det? Jo, eftersom $\phi_i(x)$ är noll utanför intervallet $[x_{i-1}, x_{i+1}]$ så kommer alla produkter $\phi_k'\phi_i'$, $\phi_k'\phi_i$ och $\phi_k\phi_i$ i (18) att bli identiskt noll utom om k är lika med i eller är närmaste granne, dvs $k = i-1$ eller $k = i+1$. I den i-te ekvationen kommer därför endast elementen $a_{i,i-1}$, a_{ii} och $a_{i,i+1}$ att vara skilda från noll, och ekvation (19) reduceras till

$$a_{i,i-1}c_{i-1} + a_{ii}c_i + a_{i,i+1}c_{i+1} = f_i. \tag{20}$$

Vi studerar matriselementen lite närmare. Elementet $a_{i,i-1}$ innehåller hattfunktionsprodukter som är skilda från noll enbart i intervallet $[x_{i-1}, x_i]$:

$$a_{i,i-1} = -\int_{x_{i-1}}^{x_i} \phi'_{i-1}\phi'_i dx + \int_{x_{i-1}}^{x_i} p\,\phi'_{i-1}\phi_i dx + \int_{x_{i-1}}^{x_i} q\,\phi_{i-1}\phi_i dx =$$

$$= -\int_{x_{i-1}}^{x_i} \frac{-1}{h_{i-1}}\frac{1}{h_{i-1}} dx + \int_{x_{i-1}}^{x_i} p(x)\frac{-1}{h_{i-1}}\frac{(x-x_{i-1})}{h_{i-1}} dx +$$

$$+ \int_{x_{i-1}}^{x_i} q(x)\frac{(x_i-x)}{h_{i-1}}\frac{(x-x_{i-1})}{h_{i-1}} dx =$$

$$= \frac{1}{h_{i-1}^2}\left(h_{i-1} - \int_{x_{i-1}}^{x_i} p(x)(x-x_{i-1})dx + \int_{x_{i-1}}^{x_i} q(x)(x_i-x)(x-x_{i-1})dx\right)$$

För elementet a_{ii} sträcker sig integrationsintervallet över $[x_{i-1}, x_{i+1}]$:

$$a_{ii} = -\int_{x_{i-1}}^{x_{i+1}} {\phi'_i}^2 dx + \int_{x_{i-1}}^{x_{i+1}} p\,\phi'_i\phi_i dx + \int_{x_{i-1}}^{x_{i+1}} q\,\phi_i^2 dx =$$

$$= \frac{1}{h_{i-1}^2}\left(-h_{i-1} + \int_{x_{i-1}}^{x_i} p(x)(x-x_{i-1})dx + \int_{x_{i-1}}^{x_i} q(x)(x-x_{i-1})^2 dx\right) +$$

$$+ \frac{1}{h_i^2}\left(-h_i - \int_{x_i}^{x_{i+1}} p(x)(x_{i+1}-x)dx + \int_{x_i}^{x_{i+1}} q(x)(x_{i+1}-x)^2 dx\right)$$

Elementet $a_{i,i+1}$ kräver integration över intervallet $[x_i, x_{i+1}]$ enligt:

$$a_{i,i+1} = -\int_{x_i}^{x_{i+1}} \phi'_{i+1}\phi'_i dx + \int_{x_i}^{x_{i+1}} p\,\phi'_{i+1}\phi_i dx + \int_{x_i}^{x_{i+1}} q\,\phi_{i+1}\phi_i dx =$$

$$= \frac{1}{h_i^2}\left(h_i + \int_{x_i}^{x_{i+1}} p(x)(x_{i+1}-x)dx + \int_{x_i}^{x_{i+1}} q(x)(x-x_i)(x_{i+1}-x)dx\right)$$

Matriselementen kan fomuleras mer kompakt med hjälp av beteckningarna

$$P_{1,i} = \int_{x_{i-1}}^{x_i} p(x)(x-x_{i-1})dx, \quad P_{2,i} = \int_{x_i}^{x_{i+1}} p(x)(x_{i+1}-x)dx,$$

$$Q_{1,i} = \int_{x_{i-1}}^{x_i} q(x)(x-x_{i-1})^2 dx, \quad Q_{2,i} = \int_{x_i}^{x_{i+1}} q(x)(x_{i+1}-x)^2 dx,$$

$$S_i = \int_{x_{i-1}}^{x_i} q(x)(x_i-x)(x-x_{i-1})dx$$

De tre matriselementen och högerledet i ekvation (20) blir därmed

$$\begin{array}{ll} a_{i,i-1} & = (h_{i-1} - P_{1,i} + S_i)/h_{i-1}^2 \\ a_{ii} & = (-h_{i-1} + P_{1,i} + Q_{1,i})/h_{i-1}^2 + (-h_i - P_{2,i} + Q_{2,i})/h_i^2 \\ a_{i,i+1} & = (h_i + P_{2,i} + S_{i+1})/h_i^2 \\ f_i & = \frac{1}{h_{i-1}} \int_{x_{i-1}}^{x_i} f(x)(x - x_{i-1})dx + \frac{1}{h_i} \int_{x_i}^{x_{i+1}} f(x)(x_{i+1} - x)dx. \end{array} \tag{21}$$

Om funktionerna $p(x)$, $q(x)$ och $f(x)$ har enkla uttryck kan vi integrera analytiskt, annars måste vi ta till numerisk integration.

7.9.4 Finslipning av Galerkins algoritm

Första och sista raden i ekvationssystemet $\mathbf{Ac} = \mathbf{f}$ kräver specialbehandling. För $i = 1$ lyder ekvationen $a_{1,0}\, c_0 + a_{1,1}\, c_1 + a_{1,2}\, c_2 = f_1$. Vi vet från randvillkoren (14) att $c_0 = \alpha$ och första ekvationen blir

$$a_{1,1}\, c_1 + a_{1,2}\, c_2 = f_1 - a_{1,0}\alpha.$$

För $i = n = N - 1$ lyder ekvationen $a_{n,n-1}c_{n-1} + a_{nn}c_n + a_{n,n+1}c_{n+1} = f_n$. Här är $c_{n+1} = c_N = \beta$ känd och n-te ekvationen blir därför

$$a_{n,n-1}c_{n-1} + a_{nn}c_n = f_n - a_{n,n+1}\beta.$$

Sista fasen i algoritmen är att lösa det tridiagonala systemet och erhålla koefficienterna $c_1, c_2, \ldots, c_n$. Komplettera med c_0 och c_{n+1} från (14). Därmed har vi funnit den approximativa galerkinlösningen $u(x) = \sum_{k=0}^{N} c_k \phi_k(x)$ till randvärdesproblemet.

Exempel: Vi prövar Galerkins metod på uppgift B20 i Blandade problem: $y'' + 4y'/x - 2y/x^2 = -2(\ln x)/x^2$ med randvillkoren $y(1) = 0,\ y(5) = 10$. Lösningen ska beräknas och ritas i intervallet $1 \le x \le 5$. Utskrift ska göras av $y(3)$; önskemålet är tre korrekta siffror i detta värde.

Här gäller $p(x) = 4/x,\ q(x) = -2/x^2$ och $f(x) = -2(\ln x)/x^2$. Vi gör ekvidistant indelning med steget $h = (5-1)/N$, först med $N = 10$ intervall, sedan med successiv fördubbling av antalet intervall.

Integralerna $P_{1,i}$, $P_{2,i}$, $Q_{1,i}$, $Q_{2,i}$ och S_i kan vi beräkna analytiskt:

$$P_{1,i} = 4 \int_{x_{i-1}}^{x_i} (x - x_{i-1})/x\, dx = 4(h - x_{i-1} \ln \frac{x_i}{x_{i-1}})$$

$$P_{2,i} = 4 \int_{x_i}^{x_{i+1}} (x_{i+1} - x)/x\, dx = 4(x_{i+1} \ln \frac{x_{i+1}}{x_i} - h)$$

$$Q_{1,i} = -2 \int_{x_{i-1}}^{x_i} \frac{(x - x_{i-1})^2}{x^2} dx = -2(h - 2x_{i-1} \ln \frac{x_i}{x_{i-1}} + h\frac{x_{i-1}}{x_i})$$

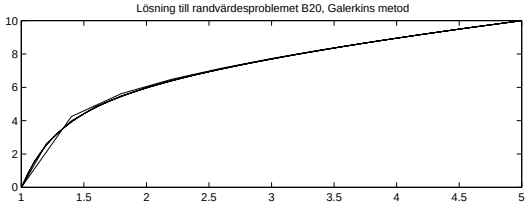
$$Q_{2,i} = -2\int_{x_i}^{x_{i+1}} \frac{(x_{i+1}-x)^2}{x^2}dx = -2(h\frac{x_{i+1}}{x_i} - 2x_{i+1}\ln\frac{x_{i+1}}{x_i} + h)$$

$$S_i = -2\int_{x_{i-1}}^{x_i} \frac{(x_i-x)(x-x_{i-1})}{x^2}dx = -2(-2h + (x_{i-1}+x_i)\ln\frac{x_i}{x_{i-1}})$$

Högerledsintegralen f_i i (21) kräver numerisk integration — här utnyttjar vi trapetsregeln med varje integrationsintervall h delat i åtta delintervall.

```
function randB20galerk   % B20 med Galerkins metod
  clear, clf
  alfa=0; beta=10;
  N=10; Ymitt=[]; subplot(2,1,1)
  for etapp=1:4
    h=4/N; n=N-1; x=1+h*(1:n)'; X=[1; x; 5];
    xminus=X(1:n); xplus=X(3:N+1);
    P1=4*(h-xminus.*log(x./xminus));
    P2=4*(xplus.*log(xplus./x)-h);
    Q1=-2*(h-2*xminus.*log(x./xminus)+h*xminus./x);
    Q2=-2*(h*xplus./x-2*xplus.*log(xplus./x)+h);
    S =-2*(-2*h+(xminus+x).*log(x./xminus));
    SN=-2*(-2*h+(x(n)+5)*log(5/x(n)));

    F=[]; dh=h/8;  % högerledsintegraler med trapetsregeln
    for i=2:N
      xL=X(i-1)+dh*(0:8); fL=-2*log(xL)./xL.^2.*(xL-X(i-1));
      xR=X(i)+dh*(0:8);   fR=-2*log(xR)./xR.^2.*(X(i+1)-xR);
      TL=dh*(sum(fL)-0.5*(fL(1)+fL(9)));
      TR=dh*(sum(fR)-0.5*(fR(1)+fR(9)));
      F=[F; 1/h*(TL+TR)];
    end

    dia=(-2*h+P1-P2+Q1+Q2)/h^2;
    sup=(h+P2(1:n-1)+S(2:n))/h^2;
    sub=(h-P1(2:n)+S(2:n))/h^2;
    F(1)=F(1)-alfa*(h-P1(1)+S(1))/h^2;
    F(n)=F(n)-beta*(h+P2(n)+SN)/h^2;
    c=tridia(dia,sup,sub,F);
    Y=[alfa; c; beta]; plot(X,Y), hold on
    Ymitt=[Ymitt Y(N/2+1)]; N=2*N;
  end, Ymitt
```

Med $N = 10, 20, 40, 80$ erhålls för $x = 3$ successivt: $y_{mitt} =$ 7.7277, 7.7035, 7.6974, 7.6959.



7.10 Kollokation med bellsplines

Vi betraktar samma randvärdesproblem som tidigare, alltså (12):

$$y'' + p(x)\,y' + q(x)\,y = f(x), \quad a \le x \le b, \quad y(a) = \alpha, \;\; y(b) = \beta.$$

Nu ska vi angripa problemet med metoden kollokation. Liksom vid Galerkins metod ansätter man $y(x)$ som en linjärkombination av basfunktioner

$$y(x) \approx \sum c_k \phi_k(x).$$

Men i stället för ortogonalitetsvillkor som galerkinmetoden bygger på, finns *interpolationsvillkor* i kollokationsmetoden. Residualfunktionen ska anta värdet noll i ett antal interpolationspunkter.

De valda basfunktionerna måste nu vara två gånger kontinuerligt deriverbara; de ska nämligen direkt kunna stoppas in i differentialekvationen. Linjära splines – hattfunktionerna – duger inte längre, däremot är de kubiska bellsplinefunktionerna $B_k(x)$ som presenterades i avsnitt 5.2 ett gott val av basfunktioner. Kubiska splines har önskade derivataegenskaper, $B_k'(x)$ och $B_k''(x)$ är kontinuerliga överallt.

I differentialekvationen ovan approximerar vi nu $y(x)$ med $\sum_{k=0}^{n+1} c_k B_k(x)$, derivatan $y'(x)$ av $\sum_{k=0}^{n+1} c_k B_k'(x)$ och $y''(x)$ av $\sum_{k=0}^{n+1} c_k B_k''(x)$.

Varför summeringen går från 0 till $n{+}1$ kommer att framgå av de fortsatta beräkningarna.

Efter insättning erhålls

$$\sum_{k=0}^{n+1} c_k \left(B_k''(x) + p(x)B_k'(x) + q(x)B_k(x)\right) \approx f(x). \tag{22}$$

7.10.1 Kollokationsmetodens interpolationsvillkor

För att kunna bestämma de okända koefficienterna $c_0, c_1, \ldots, c_{n+1}$ måste vi ha tillgång till $n + 2$ ekvationer. Två samband kommer från de kända randvärdena $y(a) = \alpha$ och $y(b) = \beta$ och leder till

$$\sum_{k=0}^{n+1} c_k B_k(a) = \alpha, \quad \sum_{k=0}^{n+1} c_k B_k(b) = \beta. \tag{23}$$

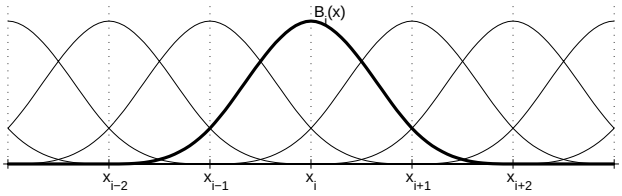
Vid kollokation utnyttjas interpolation i n punkter för att få de n sambanden som återstår. Man kan plocka ut interpolationspunkter på många olika sätt, men enklast är att låta dem sammanfalla med bellsplineuppsättningens

knutpunkter som i sin tur placeras ekvidistant i intervallet $a \le x \le b$ med $x_1 = a,\ x_2 = a+h, \ldots,\ x_n = b$, där $h = (b-a)/(n-1)$. När knutpunkterna ligger på lika avstånd kan vi utnyttja bellsplinefunktionerna i avsnitt 5.2.3. (Här ändrar vi dock beteckningen T_i till x_i.)

7.10.2 Egenskaper hos ekvidistanta bellsplines

Varje bellspline $B_i(x)$ är noll utanför området $x_{i-2} \le x \le x_{i+2}$. Eftersom uttrycket (22) är uppbyggt av basfunktionerna $B_0(x), \ldots, B_{n+1}(x)$, behövs förutom de givna knutpunkterna tre stycken till vänster om x_1 placerade vid $x_0 = x_1 - h,\ x_{-1} = x_1 - 2h,\ x_{-2} = x_1 - 3h,$ och tre bortom högra randen: $x_{n+1} = x_n + h,\ x_{n+2} = x_n + 2h,\ x_{n+3} = x_n + 3h$.

I de fyra intervall där den klockformade bellsplinekurvan är skild från noll gäller följande uttryck för $B_i(x)$ och dess derivator:



$$x_{i-2} \le x \le x_{i-1}: \quad t = \tfrac{1}{h}(x - x_{i-2}), \quad B_i(x) = \tfrac{1}{6}t^3, \ \ B_i'(x) = \tfrac{1}{2h}t^2, \ \ B_i''(x) = \tfrac{1}{h^2}t$$

$$x_{i-1} \le x \le x_i: \quad t = \tfrac{1}{h}(x - x_{i-1}), \quad \begin{array}{l} B_i(x) = \frac{1}{6} + \frac{1}{2}(t + t^2 - t^3) \\ B_i'(x) = \frac{1}{2h}(1 + 2t - 3t^2) \\ B_i''(x) = \frac{1}{h^2}(1 - 3t) \end{array}$$

$$x_i \le x \le x_{i+1}: \quad t = \tfrac{1}{h}(x_{i+1} - x), \quad \begin{array}{l} B_i(x) = \frac{1}{6} + \frac{1}{2}(t + t^2 - t^3) \\ B_i'(x) = -\frac{1}{2h}(1 + 2t - 3t^2) \\ B_i''(x) = \frac{1}{h^2}(1 - 3t) \end{array}$$

$$x_{i+1} \le x \le x_{i+2}: \quad t = \tfrac{1}{h}(x_{i+2} - x), \quad B_i(x) = \tfrac{1}{6}t^3, \ \ B_i'(x) = -\tfrac{1}{2h}t^2, \ \ B_i''(x) = \tfrac{1}{h^2}t$$

Ur figuren och uttrycken ovan kan vi konstatera att vid varje knutpunkt x_i finns bara tre bellsplinekurvor skilda från noll:

$$\begin{array}{lll} B_{i-1}(x_i) = 1/6 & B_i(x_i) = 2/3, & B_{i+1}(x_i) = 1/6 \\ B_{i-1}'(x_i) = -1/2h & B_i'(x_i) = 0, & B_{i+1}'(x_i) = 1/2h \\ B_{i-1}''(x_i) = 1/h^2 & B_i''(x_i) = -2/h^2, & B_{i+1}''(x_i) = 1/h^2 \end{array} \tag{24}$$

Det innebär att randvärdessambanden (23) får följande uttryck – vi utnyttjar att $a = x_1$ och $b = x_n$:

$$c_0 B_0(x_1) + c_1 B_1(x_1) + c_2 B_2(x_1) = \alpha \implies c_0/6 + 2c_1/3 + c_2/6 = \alpha.$$

$$c_{n-1}B_{n-1}(x_n) + c_nB_n(x_n) + c_{n+1}B_{n+1}(x_n) = \beta \implies$$
$$c_{n-1}/6 + 2c_n/3 + c_{n+1}/6 = \beta.$$

Vi löser ut c_0 och c_{n+1} eftersom de behövs senare:

$$c_0 = 6\alpha - 4c_1 - c_2, \quad c_{n+1} = 6\beta - c_{n-1} - 4c_n. \tag{25}$$

7.10.3 Interpolationssamband

Nu återgår vi till interpolationssambanden som innebär att uttrycket (22) ska vara satisfierat (VL=HL) vid knutpunkterna x_i, $i = 1, 2, \ldots, n$:

$$\sum_{k=0}^{n+1} c_k \left(B_k''(x_i) + p(x_i)B_k'(x_i) + q(x_i)B_k(x_i)\right) = f(x_i), \quad i = 1, 2, \ldots, n.$$

Vi inför beteckningen a_{ik} för uttrycket inom parentesen i vänsterledet, alltså

$$a_{ik} = B_k''(x_i) + p(x_i)B_k'(x_i) + q(x_i)B_k(x_i).$$

Tack vare de goda egenskaperna (24) hos bellsplinefunktionerna kommer endast $a_{i,i-1}$, a_{ii} och $a_{i,i+1}$ att vara skilda från noll i den i-te ekvationen, dvs de linjära sambanden reduceras till

$$a_{i,i-1}c_{i-1} + a_{ii}c_i + a_{i,i+1}c_{i+1} = f(x_i), \quad i = 1,\, 2, \ldots, n, \tag{26}$$

där matriselementen uppfyller (visa detta):

$$a_{i,i-1} = \frac{1}{h^2} - \frac{p(x_i)}{2h} + \frac{q(x_i)}{6}, \quad a_{ii} = -\frac{2}{h^2} + \frac{2q(x_i)}{3}, \quad a_{i,i+1} = \frac{1}{h^2} + \frac{p(x_i)}{2h} + \frac{q(x_i)}{6}.$$

Med utnyttjande av randvärdessambanden (25) bildar de n ekvationerna (26) ett tridiagonalt ekvationssystem.

7.10.4 Finslipning av algoritmen för kollokation

Som vanligt vid randvärdesproblem måste man ägna särskild möda åt första och sista raden i ekvationssystemet. I första ekvationen (för $i = 1$) ingår c_0 som med hjälp av (25) kan elimineras:

$$a_{1,0}(6\alpha - 4c_1 - c_2) + a_{1,1}c_1 + a_{1,2}c_2 = f(x_1).$$

Lite omstuvning leder till

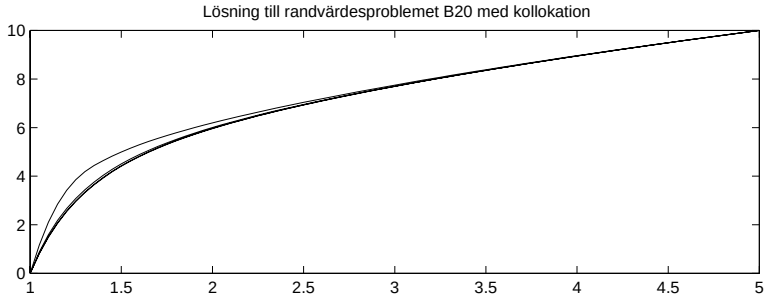
$$(a_{1,1} - 4a_{1,0})c_1 + (a_{1,2} - a_{1,0})c_2 = f(x_1) - 6a_{1,0}\alpha.$$

Vid $i = n$: $a_{n,n-1}c_{n-1} + a_{n,n}c_n + a_{n,n+1}(6\beta - c_{n-1} - 4c_n) = f(x_n) \implies$

$$(a_{n,n-1} - a_{n,n+1})c_{n-1} + (a_{n,n} - 4a_{n,n+1})c_n = f(x_n) - 6a_{n,n+1}\beta.$$

Därmed har vi lyckats ställa upp ett tridiagonalt system för bestämning av $c_1, \ldots, c_n$. När dessa koefficienter beräknats erhålls c_0 och c_{n+1} ur (25) och kollokationsproblemet är löst. Slutligen bör förstås differentialekvationslösningen $\sum_{k=0}^{n+1} c_k B_k(x)$ ritas upp.

Exempel: Vi prövar att lösa randvärdesproblemet B20 i Blandade problem med kollokation. Test görs med 10, 20, 40 och 80 ekvidistanta delintervall (11, 21, 41, 81 knutpunkter). Funktionen `fbelleq` som används i uppritningen presenterades i avsnitt 5.2.3.




```
  clear, clf
  alfa=0; beta=10;
  N=10; Ymitt=[];
  for etapp=1:4                          % N = 10, 20, 40, 80 intervall
    n=N+1; h=4/N; x=(1:h:5)';            % n = antal interpol.punkter
    p=4./x; q=-2./x.^2; f=-2*log(x)./x.^2;
    dia=-2/h^2+2*q/3;
    sup=1/h^2+p/(2*h)+q/6;
    sub=1/h^2-p/(2*h)+q/6;
    a_10=sub(1); sub=sub(2:n);
    a_ns=sup(n); sup=sup(1:n-1);
    dia(1)=dia(1)-4*a_10; sup(1)=sup(1)-a_10;     f(1)=f(1)-6*a_10*alfa;
    dia(n)=dia(n)-4*a_ns; sub(n-1)=sub(n-1)-a_ns; f(n)=f(n)-6*a_ns*beta;
    c=tridia(dia,sup,sub,f);
    C=[6*alfa-4*c(1)-c(2); c; 6*beta-c(n-1)-4*c(n)];
    X=(1:0.05:5)';
    Y=0; for k=1:n+2, Y=Y+C(k)*fbelleq(k-1,X,1,5,n); end
    subplot(2,1,1), plot(X,Y), hold on
    title('Lösning till randvärdesproblemet B20 med kollokation')
    Ymitt=[Ymitt Y((length(X)-1)/2+1)]; N=2*N;
  end, Ymitt
% Resultat: N=10, 20, 40, 80 ger  ymitt = 7.7492, 7.7071, 7.6982, 7.6961
```

8 Begynnelsevärdesproblem för ODE

8.1 Allmänt om numerisk lösning av ODE

Friska upp kunskaperna om ordinära differentialekvationer och grundläggande numerisk behandling av begynnelsevärdesproblem för ODE genom att läsa NAM avsnitt 8.1 – 8.6.

Urtypen för stegmetoder för numerisk lösning av $y' = f(t, y)$ med givet begynnelsvillkor är Eulers metod, $y_{j+1} = y_j + h\, f(t_j, y_j)$, som bygger på framåtdifferenskvot som approximation till derivatan. Trunkeringsfelet är $O(h)$. En annan välkänd standardmetod är Runge-Kuttas fjärdeordningsmetod (RK4) med trunkeringsfelet $O(h^4)$ och med formeln

$$y_{j+1} = y_j + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4), \quad \text{där} \quad \begin{cases} f_1 = f(t_j,\, y_j) \\ f_2 = f(t_j{+}h/2,\, y_j{+}hf_1/2) \\ f_3 = f(t_j{+}h/2,\, y_j{+}hf_2/2) \\ f_4 = f(t_j{+}h,\, y_j{+}hf_3) \end{cases}$$

Eulers metod och Runge-Kuttas metod är exempel på explicita enstegsmetoder. Vi ska i det här kapitlet också presentera två implicita enstegsmetoder, bakåteulermetoden (eller implicit Euler), $y_{j+1} = y_j + h\, f(t_{j+1}, y_{j+1})$, och trapetsmetoden $y_{j+1} = y_j + \frac{h}{2}(f(t_j, y_j) + f(t_{j+1}, y_{j+1}))$.
De karakteriseras av att den obekanta storheten y_{j+1} även finns i högerledet. Någon form av ekvationslösning måste därför tillämpas för att bestämma y_{j+1}-värdet.

De nämnda explicita och implicita metoderna är grundprincipen för mer sofistikerade numeriska algoritmer för ordinära differentialekvationer. I MATLAB finns stegreglerande algoritmer såsom `ode23, ode45, ode15s`, med flera (`help ode45` ger information).

8.2 Numerisk stabilitet och styva ODE-problem

Tre saker bör beaktas vid valet av numerisk stegmetod: enkelhet, noggrannhet, stabilitet. När det gäller enkelhet vinner Eulers metod. Noggrannhetsmässigt kommer Eulers metod och bakåteuler på delad jumboplats med trunkeringsfel $O(h)$. Här är RK4 och `ode45` betydligt bättre med trunkeringsfel $O(h^4)$. Den tredje aspekten gäller stabilitet, och här visar det sig att de explicita metoderna i vissa fall är värdelösa, medan bakåteuler och trapetsmetoden är vinnare.

Studera Heath hela kap 9.

8.2.1 Stabilitetsområde för Eulers metod och för bakåteuler

Låt oss betrakta följande enkla differentialekvation (som brukar benämnas testekvation): $u' = q\,u,\ u(0) = u_0,$ vars sanna lösning är $u(t) = u_0 e^{q\,t}$.

När Eulers metod tillämpas får vi differensekvationen $u_{j+1} = u_j + \delta t\, q\, u_j$ som också kan skrivas $u_{j+1} = (1 + q\,\delta t)u_j = G\,u_j,$ vilket innebär att värdet u_{j+1} vid nästa tidssteg är faktorn $G = 1 + q\,\delta t$ multiplicerad med det föregående värdet u_j. Den numeriska lösningen kan då skrivas $u_j = (1+q\,\delta t)^j u_0$.

Om q är ett reellt och negativt tal går den exakta lösningen $u_0 e^{q\,t}$ monotont mot noll då t växer. Den numeriska metodens successiva värden måste ha samma egenskap. Faktorn G måste alltså vara till beloppet mindre än ett för att Eulers metod ska vara en stabil metod vid negativt q-värde.

Kravet är alltså $|1+q\,\delta t| < 1,$ vilket också kan skrivas $-1 < 1+q\,\delta t < 1$. För negativt q är den högra olikheten alltid uppfylld. Den vänstra olikheten kan skrivas $-q\,\delta t < 2$. Kravet för att Eulers metod över huvudtaget ska vara användbar är därför att tidssteget δt måste uppfylla

$$0 < \delta t < \frac{2}{(-q)}.$$

Man säger att stabilitetströskeln är $\delta t = 2/(-q)$. Om vi väljer ett tidssteg större än stabilitetströskeln blir u_{j+1}-värdet till beloppet större än föregående värde u_j. Lösningen blir instabil med obehagligt växande oscillationer, se exemplet nedan.

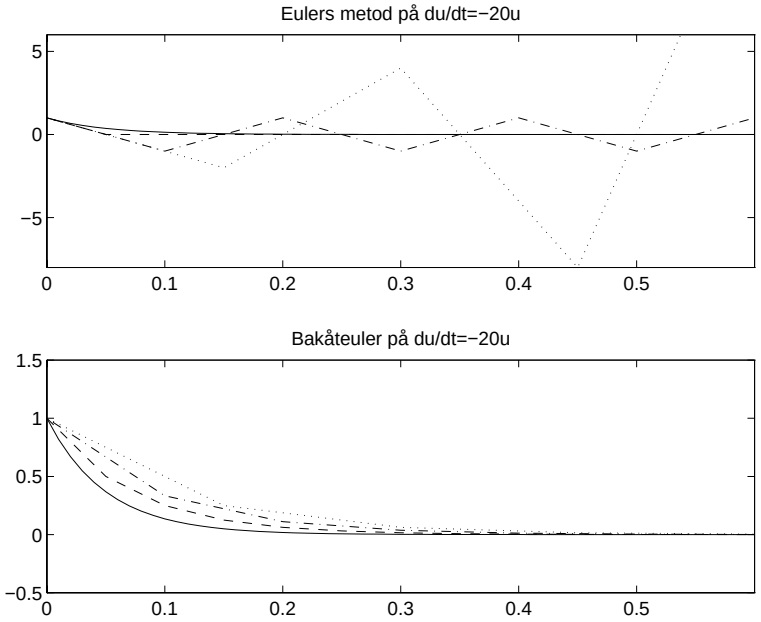
Nu studerar vi bakåteulermetoden, $u_{j+1} = u_j + \delta t\, f(t_{j+1}, u_{j+1}),$ på samma differentialekvation $u' = q\,u,\ u(0) = u_0$. Det ger $u_{j+1} = u_j + \delta t\, q\, u_{j+1},$ vilket kan skrivas om som $(1 - q\,\delta t)u_{j+1} = u_j,$ eller $u_{j+1} = u_j/(1 - q\,\delta t)$. När q är negativt är nämnaren större än ett och kravet $|u_{j+1}| < |u_j|$ är alltid uppfyllt oberoende av hur vi väljer tidssteget δt.

Bakåteulermetoden är alltså en stabil metod för alla tidssteg δt, och det blir i stället noggrannhetskravet på lösningen som får vägleda oss i lämpligt val av steglängd.

Exempel: Vi prövar Eulers metod på $u' = -20u,\quad u(0) = 1$. I första försöket använder vi $\delta t = 0.15$. Därefter gör vi om eulerberäkningarna med $\delta t = 0.1$ och slutligen med 0.05. Med $\delta t = 0.15$ blir de successivt erhållna u-värdena -2, 4, -8, 16, ..., eftersom faktorn $G = 1+q\,\delta t$ är $1-20\cdot 0.15 = -2$. Mycket instabilt alltså! Resultatet är prickat i figuren.

Med $\delta t = 0.1$ gäller $G = 1 - 20 \cdot 0.1 = -1,$ och u-värdena sicksackar mellan -1 och 1, vilket ju inte heller stämmer med det verkliga förloppet.

Med $\delta t = 0.05$ får vi $G = 1 - 20 \cdot 0.05 = 0$; det innebär att den numeriska lösningen redan efter ett tidssteg hamnar på noll och stannar kvar där. Det är ju stabilt men inte helt korrekt. Den exakta lösningen $u(t) = e^{-20t}$ är heldragen, från startvärdet ett avklingar den monotont mot noll.



Samma uppsättning tidssteg har valts i bakåteulermetoden. Resultatet med steget $\delta t = 0.15$ är prickat i figuren ovan, med $\delta t = 0.1$ streckprickat och med $\delta t = 0.05$ streckat. Den exakta differentialekvationslösningen $u(t) = e^{-20t}$ är den heldragna kurvan. Bakåteuler är stabil för alla val av δt (inga växande oscillationer uppstår), men som synes bör man ur noggrannhetssynpunkt inte välja tidssteget för stort.

8.2.2 Styva differentialekvationsproblem

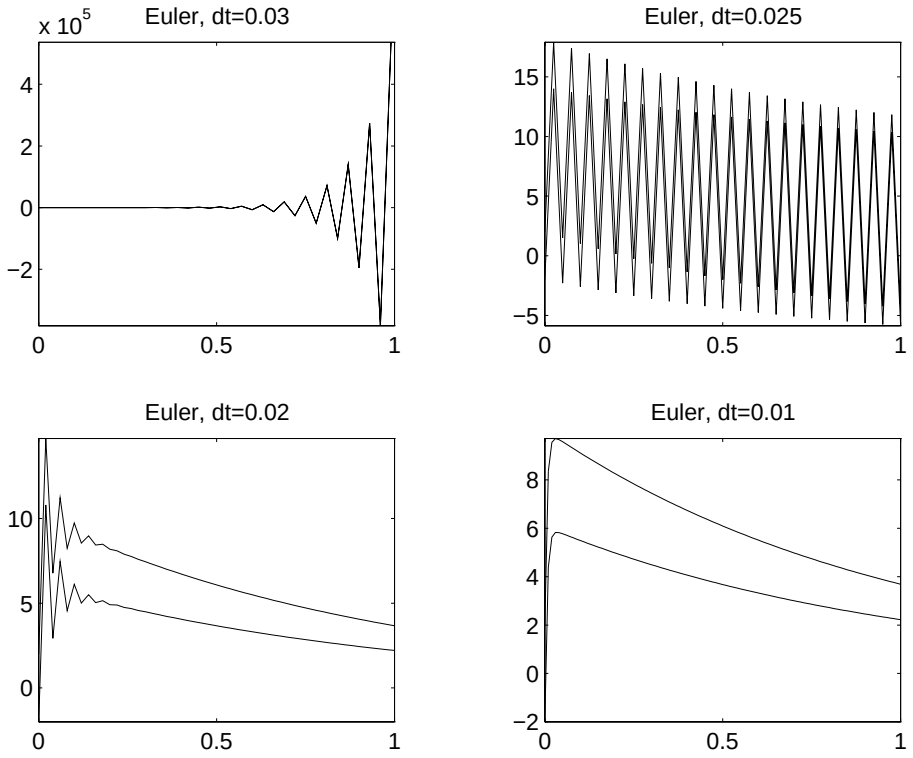
Ett litet system av linjära differentialekvationer är givet:

$$\begin{cases} du_1/dt = a_1 u_1 + a_2 u_2 \\ du_2/dt = (a_1+1)u_1 + (a_2-1)u_2 \end{cases} \quad \text{med } u_1(0) = -2, \;\; u_2(0) = 2.$$

Koefficienterna är $a_1 = -200$ och $a_2 = 120$. Vi vill finna en stabil numerisk algoritm för att beräkna lösningskurvorna för $u_1(t)$ och $u_2(t)$ fram till $t = 1$.

Differentialekvationssystemet kan skrivas på formen

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\,\mathbf{u} \quad \text{där } \mathbf{A} = \begin{pmatrix} a_1 & a_2 \\ a_1+1 & a_2-1 \end{pmatrix} = \begin{pmatrix} -200 & 120 \\ -199 & 119 \end{pmatrix}$$



Detta differentialekvationssystem kan lösas analytiskt och den exakta lösningen lyder $u_1(t) = \frac{480}{79}e^{-t} - \frac{638}{79}e^{-80t}$ och $u_2(t) = \frac{796}{79}e^{-t} - \frac{638}{79}e^{-80t}$. Både $u_1(t)$ och $u_2(t)$ innehåller dels en långsamt avtagande exponentialfunktion, dels en mycket snabbt avklingande, nämligen e^{-80t}. Detta är vad som karakteriserar ett så kallat *styvt* differentialekvationsproblem; lösningen är en kombination av snabba och långsamma förlopp.

Eulers metod på diffekvationssystemet leder till $\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + \delta t\, \mathbf{A}\, \mathbf{u}^{(j)}$. Vi prövar med steglängden $\delta t = 0.03$. Resultatet syns i övre vänstra figuren. Det går alldeles åt skogen, skalan på y-axeln har fått enheten 10^5, lösningen sicksackar med kraftigt växande amplitud. Med steglängden 0.025 är det också sicksackigt (övre högra figuren) men lösningen håller sig begränsad. Steget 0.02 ger oscillationer i början men de planar ut så småningom till en stabil lösningskurva. Man får minska steget δt ytterligare för att få en helt acceptabel numerisk lösning till problemet. Eulers metod är alltså instabil om steget väljs för stort.

Det finns en stabilitetströskel för tidssteget även här. För ett linjärt

differentialekvationssystem $\frac{d\mathbf{u}}{dt} = \mathbf{A}\,\mathbf{u} + \mathbf{b}$, spelar storleken på egenvärdena till $\mathbf{A}$ en stor roll; det är det mest negativa egenvärdet som sätter tröskeln för tidssteget. När Eulers metod används för den numeriska lösningen bestäms stabilitetströskeln av

$$\delta t < \frac{2}{\max(-\lambda(\mathbf{A}))}$$

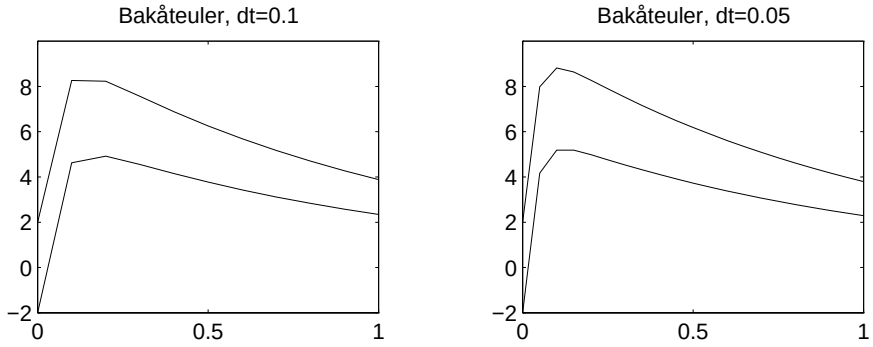
Matrisen $\mathbf{A}$ i vårt exempel har egenvärdena -80 och -1; det innebär att steget måste vara mindre än $2/80 = 0.025$ för att Eulers metod ska bli stabil. Det var vad vi kunde konstatera genom experimenten också. (Egenvärdena -80 och -1 hör ihop med den exakta lösningens båda termer e^{-80t} resp e^{-t}.)

När matrisens egenvärden är negativa och av mycket olika storlek utgör systemet ett styvt differentialekvationsproblem. Om en explicit metod utnyttjas är man tvungen att rätta steglängden efter det snabbaste förloppet. Detta är dilemmat för styva problem: det är den mest stabila lösningskomponenten — den som snabbast dör ut — som begränsar steglängden.

Vid styva problem är inte heller den explicita metoden RK4 lämplig utan bäst är att utnyttja en implicit metod som inte spårar ur på grund av stabilitetskrångel, till exempel bakåteulermetoden eller trapetsmetoden (eller MATLABs `ode23s, ode15s`).

Bakåteulermetoden på differentialekvationssystemet $\mathbf{u}' = \mathbf{A}\,\mathbf{u}$ lyder $\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + \delta t\,\mathbf{A}\,\mathbf{u}^{(j+1)}$. Omskrivning leder till $(\mathbf{I} - \delta t\,\mathbf{A})\,\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)}$. Det är ett linjärt ekvationssystem med känd systemmatris i vänsterledet och det kan lätt lösas i MATLAB.

Det fina med bakåteulermetoden är att den aldrig någonsin ger upphov till instabilitet med växande sicksackfenomen. Alla steglängdsval är tillåtna, men naturligtvis ska steget väljas så litet att lösningen får acceptabel precision.



```
% styveul, styvt system med Euler och bakåteuler
  clear, figure(1), clf
  ustart=[-2 2]';  tslut=1; a1=-200; a2=120;  A=[a1  a2;  a1+1  a2-1];
  dtkoll=[0.03 0.025 0.02 0.01];
  for nr=1:4
    dt=dtkoll(nr); u=ustart;  t=0;  U=u';  T=t;
    while t<tslut-dt/2
      u=u+dt*A*u; t=t+dt;  U=[U; u'];  T=[T; t];
    end
    subplot(2,2,nr), plot(T,U), title(['Euler, dt=' num2str(dt)])
  end
% Bakåteuler, betydligt större steg kan användas
  dtkoll=[0.1 0.05];
  figure(2), clf
  for nr=1:2
    dt=dtkoll(nr); u=ustart;  t=0; U=u'; T=t;
    C=eye(2)-dt*A;
    while t<tslut-dt/2
      u=C\u;  t=t+dt;  U=[U; u']; T=[T; t];
    end
    subplot(2,2,nr), plot(T,U), title(['Bakåteuler, dt=' num2str(dt)])
  end
```

En implicit metod som liknar bakåteulermetoden är trapetsmetoden:

$$y_{j+1} = y_j + \frac{\delta t}{2}(f(t_j, y_j) + f(t_{j+1}, y_{j+1})).$$

Den kan tolkas som medelvärdesbildning av Eulers metod och bakåteulermetoden, och den påminner ju starkt om trapetsregeln för numerisk integration. Trapetsmetoden är liksom bakåteulermetoden stabil för alla val av steglängder. Noggrannhetsmässigt är den fördelaktigare än både Eulers metod och bakåteuler; trapetsmetodens trunkeringsfel är proportionellt mot kvadraten på tidssteget.

Vid partiella differentialekvationer med både rums- och tidsberoende såsom värmeledningsproblem och diffusionsproblem får man i den numeriska behandlingen stor användning av trapetsmetoden, som då går under namnet Crank-Nicolsons metod.

Vi har i det här kapitlet begränsat oss till styva *linjära* differentialekvationsproblem, men resonemanget kan utsträckas till ickelinjära problem, som vi kommer att studera i samband med ickelinjära paraboliska partiella differentialekvationer senare.

9 Paraboliska partiella differentialekvationer

9.1 Partiella differentialekvationer (PDE) — grundtyper

Partiella differentialekvationer finns av tre typer: paraboliska, hyperboliska och elliptiska PDE. Det enklaste modellproblemet för vardera typen är

- Parabolisk:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le 1, \;\; 0 \le t \le t_{slut}$$

  Ett begynnelsevillkor, $u(x,0) = g(x)$ (känd funktion av x).
  Två randvillkor, $u(0,t) = f_0(t)$, $u(1,t) = f_1(t)$ ($f_0(t)$ och $f_1(t)$ kända).
  Diffusion och värmeledning är exempel på paraboliska PDE.

- Hyperbolisk:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le 1, \;\; 0 \le t \le t_{slut}$$

  Två begynnelsevillkor, $u(x,0) = g(x)$, $\frac{\partial u}{\partial t}(x,0) = 0$.
  Två randvillkor, $u(0,t) = f_0(t)$, $u(1,t) = f_1(t)$.
  Vågekvationen är den mest kända hyperboliska differentialekvationen.

- Elliptisk:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad 0 \le x \le L_x, \;\; 0 \le y \le L_y$$

  Randvillkor runt hela området: $u(0,y) = f_0(y)$, $u(L_x,y) = f_1(y)$,
  $u(x,0) = g_0(x)$, $u(x,L_y) = g_1(x)$.
  Laplaces ekvation och Poissons ekvation är exempel på elliptiska PDE.
  I elliptiska differentialekvationer finns inget tidsberoende.

Karakterisering: En andra ordningens linjär PDE i två oberoende variabler med den allmänna formen

$$a\,\frac{\partial^2 u}{\partial x^2} + b\,\frac{\partial^2 u}{\partial x \partial y} + c\,\frac{\partial^2 u}{\partial y^2} + d\,\frac{\partial u}{\partial x} + e\,\frac{\partial u}{\partial y} + f\,u + g(x,y) = 0$$

är hyperbolisk om $b^2 - 4ac > 0$, parabolisk om $b^2 - 4ac = 0$ och elliptisk om $b^2 - 4ac < 0$.

Förutom de ovan nämnda andra ordningens PDE finns det en rätt vanligt förekommande PDE av första ordningen med modellekvationen

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0$$

som för sin lösning kräver ett begynnelsevillkor och ett randvillkor. Denna PDE är närmast besläktad med den hyperboliska differentialekvationen. Lösningen $u(x,t)$ har vågkaraktär och lösningsegenskapen kallas advektion.

9.2 Numerisk behandling av paraboliska PDE

Värmeledningsekvationen

$$\frac{\partial u}{\partial t} = \beta \frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le L, \;\; 0 \le t \le t_{slut} \tag{27}$$

beskriver temperaturvariationen $u(x,t)$ i en homogen vägg med tjockleken L och värmediffusiviteten β. Diffusiviteten beror av väggmaterialets värmekonduktivitet κ, värmekapacitet c och densitet ρ enligt $\beta = \kappa/\rho c$.

Temperaturen på vardera sidan om väggen är känd, det innebär att vi har de två nödvändiga randvillkoren: $u(0,t) = f_0(t)$, $u(L,t) = f_L(t)$. Dessutom måste vi känna till temperaturen i väggtvärsnittet $0 \le x \le L$ i startögonblicket vid $t = 0$.

9.2.1 Finitadifferensmetoden, ”method of lines”

För den numeriska behandlingen använder vi finitadifferensmetoden när det gäller rumsvariablerna (i detta endimensionella fall bara variabeln x) och låter tidsberoendet finnas kvar i derivataform. Detta förfarande går under benämningen *method of lines* och leder fram till ett system av ODE som kan lösas med kända numeriska metoder för ordinära differentialekvationer.

Gör alltså en diskretisering av intervallet $0 \le x \le L$ i N delintervall med steglängden $h = L/N$. Vid varje gitterpunkt $x = x_i$ ansätter vi en obekant temperaturfunktion $u_i(t)$. Eftersom temperaturen är känd vid $x = 0$ och vid $x = L$, kommer den första obekanta temperaturfunktionen $u_1(t)$ att finnas vid $x = x_1 = h$ och den sista obekanta $u_n(t)$ vid $x = L - h$ som betecknas x_n. Antalet obekanta temperaturfunktioner blir alltså $n = N - 1$.

I värmeledningsekvationen (27) ersätter vi högerledets rumsderivator med differenskvoter:

$$\frac{du_i}{dt} = \beta \frac{u_{i+1}(t) - 2u_i(t) + u_{i-1}(t)}{h^2}, \quad i = 1, 2, ..., n. \tag{28}$$

Diskretiseringsfelet är $O(h^2)$. Med många diskretiseringspunkter, dvs med ett litet steg h, kommer FDM-lösningen förhoppningsvis att ge en god approximation till differentialekvationslösningen.

Fallen $i = 1$ och $i = n$ måste specialbehandlas eftersom de berörs av randvillkoren. Vi får ett system av differentialekvationer enligt följande, där β/h^2 betecknas γ.

$$\begin{cases} \frac{du_1}{dt} &= \gamma\,(-2u_1(t) + u_2(t) + f_0(t)) \\ \frac{du_2}{dt} &= \gamma\,(u_1(t) - 2u_2(t) + u_3(t)) \\ \frac{du_3}{dt} &= \gamma\,(u_2(t) - 2u_3(t) + u_4(t)) \\ \cdots &= \quad \cdots \\ \frac{du_n}{dt} &= \gamma\,(u_{n-1}(t) - 2u_n(t) + f_L(t)) \end{cases}$$

Differentialekvationssystemet kan skrivas i matrisform

$$\begin{pmatrix} du_1/dt \\ du_2/dt \\ du_3/dt \\ \vdots \\ du_n/dt \end{pmatrix} = \gamma \begin{pmatrix} -2 & 1 & 0 & 0 & \cdots \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & \cdots & 1 & -2 \end{pmatrix} \begin{pmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \\ \vdots \\ u_n(t) \end{pmatrix} + \gamma \begin{pmatrix} f_0(t) \\ 0 \\ 0 \\ \vdots \\ f_L(t) \end{pmatrix}$$

Låt $\mathbf{u}(t)$ vara en vektor som innehåller de okända temperaturfunktionerna $u_1(t), \ldots, u_n(t)$ och låt $\mathbf{b}(t)$ vara en vektor med de bekanta randtemperaturuttrycken $f_0(t)$ och $f_L(t)$ som första och sista komponenter och nollkomponenter där emellan. Differentialekvationssystemet kan då skrivas

$$\frac{d\mathbf{u}}{dt} = \gamma\,(\mathbf{A}\mathbf{u} + \mathbf{b}(t)).$$

9.2.2 Eulers metod, bakåteuler och Crank-Nicolsons metod

Tänkbara numeriska lösningsmetoder för differentialekvationssystemet

$$\frac{d\mathbf{u}}{dt} = \gamma\,(\mathbf{A}\mathbf{u} + \mathbf{b}(t)), \quad \mathbf{u}(0) = \mathbf{u}_{start} \tag{29}$$

är dels explicita metoder såsom Eulers metod, Runge-Kuttas metod (RK4), dels implicita metoder såsom bakåteulermetoden och trapetsmetoden.

Eulers metod lyder

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + \gamma\,\delta t\,(\mathbf{A}\mathbf{u}^{(j)} + \mathbf{b}(t^{(j)})). \tag{30}$$

Den implicita bakåteulermetoden lyder

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + \gamma\,\delta t\,(\mathbf{A}\mathbf{u}^{(j+1)} + \mathbf{b}(t^{(j+1)})). \tag{31}$$

Trapetsmetoden kallas i PDE-sammanhang ofta Crank-Nicolsons metod och har formeln

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + \frac{\gamma\,\delta t}{2}\,(\mathbf{A}\mathbf{u}^{(j)} + \mathbf{b}(t^{(j)}) + \mathbf{A}\mathbf{u}^{(j+1)} + \mathbf{b}(t^{(j+1)})). \tag{32}$$

I metoderna (31) och (32) måste vi lösa ett ekvationssystem i varje tidssteg.

Matrisen $\mathbf{A}$ som uppkommer i FDM-sammanhang har egenskapen att vara tridiagonal och diagonaltung. Alla egenvärden är negativa, men deras storlek kan variera kraftigt vilket är viktigt att tänka vid valet av lösningsmetod.

Eulers metod (30) är enklast att använda, men den är som bekant instabil om inte tidssteget δt är mycket litet. Det måste uppfylla kravet

$$\delta t < \frac{2}{\gamma \cdot \max(-\lambda_A)}$$

där egenvärdet alltså är det till beloppet största egenvärdet till matrisen $\mathbf{A}$. Alla egenvärden till $\mathbf{A}$ ligger på negativa reella axeln, och det är ofta lämpligt att utnyttja gerschgorincirklar för att finna en god skattning till det mest negativa egenvärdet.

Alla explicita stegmetoder — RK4 och `ode45`, m fl — har någon sådan begränsning för tidssteget. För RK4-metoden kan man visa att stabilitetströskeln ligger vid $\delta t = 2.7853/(\gamma\ \max(-\lambda_A))$. Större tidssteg än så leder till instabilitet.

Explicita metoder är därför i allmänhet inte praktiskt användbara vid värmeledningsproblem, eftersom det pyttelilla tidssteg som krävs för stabilitet leder till omfattande och tidskrävande beräkningar.

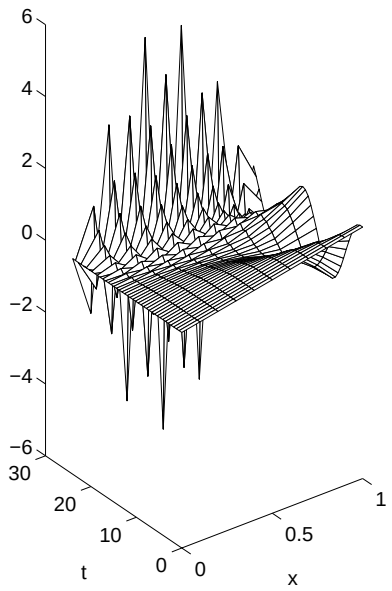
Bakåteulermetoden (31) och Crank-Nicolsons metod (32) är stabila för alla val av tidssteget δt. Programmeringsmässigt är de mycket lika. Vilken ska man då välja? Vi får titta på metodernas trunkeringsfel; bakåteuler har ett fel som är proportionelt mot δt, medan Crank-Nicolsons trunkeringsfel är proportionellt mot $(\delta t)^2$. Därför väljer vi ur noggrannhetssynpunkt helst Crank-Nicolsons metod.

Exempel: Lös värmeledningsekvationen (27) med $\beta = 0.01$, $L = 1$ och $t_{slut} = 24$. Vid $x = 0$ hålls temperaturen konstant vid noll, $u(0,t) = 0$ och vid $x = 1$ varierar temperaturen sinusformat enligt $u(1,t) = \cos 0.5t$. Begynnelsetemperaturen i väggen är $u(x,0) = x$, $0 \leq x \leq 1$.

Vi gör en diskretisering med tio delintervall i x-led så att steget h är 0.1 och $\gamma = \beta/h^2 = 0.01/0.1^2 = 1$. Eulers metod prövas för lösning av differentialekvationssystemet (29), där $\mathbf{A}$ är symmetrisk tridiagonal 9×9 matris med diagonalelementen -2 och med ettor i super- och subdiagonalen. Med hjälp av gerschgorincirklar inses att alla egenvärden finns mellan -4 och 0.
(Man kan visa att egenvärdena är $\lambda_k = -4\sin^2 \frac{k\pi}{2(n+1)}$, $k = 1, 2, ..., n$.)

Maximalt tidssteg för att Eulers metod ska vara stabil blir i detta fall $\delta t_{max} = 2/\gamma|\lambda|_{max} = 2/4 = 0.5$. Figuren visar den numeriska lösningen i ett beräkningsfall med tidssteget 0.6. (Jämför denna instabila lösning med den stabila trovärdiga temperaturfördelningen som erhålls med Crank-Nicolsons metod.)

```
% varmeul, du/dt = beta*d2u/dx2 med Euler
  beta=0.01;
  N=10; h=1/N; n=N-1;
  x=h*(1:n)'; xx=[0;x;1];
  t=0; u=x;   uu=[0; u; 1];
  T=t; U=uu';
  ett=ones(n-1,1);
  A=-2*eye(n)+diag(ett,1)+diag(ett,-1);
  gamma=beta/h^2;
  tslut=24;
  dt=input('Tidssteget dt: ');
  q=dt*gamma;  b=zeros(n,1);
  while t<tslut-dt/2
    b(n)=cos(0.5*t);
    u=u+q*(A*u+b);
    t=t+dt; T=[T; t];
    uu=[0; u; cos(0.5*t)]; U=[U; uu'];
  end
  mesh(xx',T,U)
```



Övning: Byt Eulers metod mot RK4 och experimentera med olika tidssteg. Vad blir maximalt tidssteg för stabilitet?

Vi löser nu samma problem med en implicit lösningsalgoritm och prövar Crank-Nicolsons metod. Formel (32) måste skrivas om så att alla okända $u^{(j+1)}$-termer vid tiden $t^{(j+1)}$ samlas på vänster sida. Vi inför $q = \gamma\,\delta t/2$

och får då

$$\mathbf{u}^{(j+1)} - q\,\mathbf{A}\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + q\,(\mathbf{A}\mathbf{u}^{(j)} + \mathbf{b}(t^{(j)}) + \mathbf{b}(t^{(j+1)})) \quad \Longrightarrow$$

$$(\mathbf{I} - q\mathbf{A})\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + q\,(\mathbf{A}\mathbf{u}^{(j)} + \mathbf{b}(t^{(j)}) + \mathbf{b}(t^{(j+1)})).$$

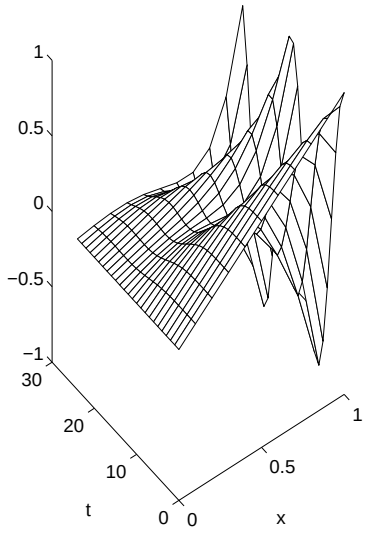
Låt den tridiagonala matrisen $\mathbf{I} - q\mathbf{A}$ betecknas med $\mathbf{C}$ och hela det kända högerledet $\mathbf{w}$. Beräkning ett tidssteg framåt med Crank-Nicolsons metod innebär lösning av ekvationssystemet $\mathbf{C}\mathbf{u}^{(j+1)} = \mathbf{w}$.

Tidssteget δt kan väljas som vi vill, inga instabiliteter uppstår. Här visas den numeriska lösningen med samma rumsindelning som i eulerberäkningen, alltså $N = 10$, och tidssteget lite större än där, $\delta t = 1$.

Om riktigt god noggrannhet önskas bör naturligtvis en finare indelning göras i x-led och ett mindre tidssteg väljas. Experimentera själv med det!

Vid stora N-värden erhålls en effektivare algoritm om det tridiagonala systemet $\mathbf{C}\mathbf{u} = \mathbf{w}$ i varje tidssteg löses med `tridia` eller ännu hellre med `sparse`-finesserna MATLAB.

```
% varmcranknic, Crank-Nicolsons metod
  beta=0.01;
  N=10; h=1/N; n=N-1;
  x=h*(1:n)'; xx=[0;x;1];
  t=0;  u=x; uu=[0; u; 1]; T=t; U=uu';
  ett=ones(n-1,1);
  A=-2*eye(n)+diag(ett,1)+diag(ett,-1);
  gamma=beta/h^2;
  tslut=24;
  dt=input('Tidssteget dt: ');
  q=dt*gamma/2;
  b=[zeros(n-1,1); cos(0.5*t)]; bp=b;
  C=eye(n)-q*A;
  while t<tslut-dt/2
    t=t+dt;  bp(n)=cos(0.5*t);
    w=u+q*(A*u+b+bp);
    u=C\w;  uu=[0; u; cos(0.5*t)];
    U=[U; uu'];  T=[T; t];
    b=bp;
  end
  mesh(xx',T,U)
```



Studera Heath avsnitt 11.1, 11.2.

9.3 Värmeledning med varierande diffusivitet

Vid diffusion och värmeledning är inte alltid diffusiviteten konstant. Om den är en funktion av x kommer differentialekvationen att få utseendet

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D(x)\frac{\partial u}{\partial x}\right), \quad 0 \le x \le L. \tag{33}$$

Om $D(x)$ är deriverbar överallt i intervallet, utför man deriveringen av högerledet och utnyttjar de vanliga centraldifferenskvoterna för $\partial^2 u/\partial x^2$ och $\partial u/\partial x$. Annars går man tillväga så här:

För högerledet i (33) använder vi följande differenskvoter som utnyttjar halvtalspunkter i punktgittret med $x_{i-0.5}$ mitt emellan x_{i-1} och x_i och med $x_{i+0.5}$ mitt emellan x_i och x_{i+1}.

$$\frac{\partial}{\partial x}\left(D(x)\,\frac{\partial u}{\partial x}\right)_{x=x_i} \approx \frac{1}{h}\left(D(x_{i+0.5})\,\frac{u_{i+1}-u_i}{h} - D(x_{i-0.5})\,\frac{u_i-u_{i-1}}{h}\right) =$$

$$= \frac{1}{h^2}\left(D(x_{i-0.5})u_{i-1} - (D(x_{i-0.5}) + D(x_{i+0.5}))u_i + D(x_{i+0.5})u_{i+1}\right).$$

Diskretiseringsfelet i differensapproximationen är $O(h^2)$.

Den paraboliska partiella differentialekvationen (33) kan därmed skrivas som ett system av ODE där den i-te differentialekvationen har uttrycket

$$\frac{du_i}{dt} = \frac{1}{h^2}\left(D_{i-0.5}u_{i-1} - (D_{i-0.5} + D_{i+0.5})u_i + D_{i+0.5}u_{i+1}\right) \tag{34}$$

för $i = 1, 2, ..., n$. När systemet skrivs på matrisformen (29) kommer vi som tidigare att erhålla en symmetrisk tridiagonal matris men nu med diagonalelementen

$(-(D_{0.5}+D_{1.5}),\ -(D_{1.5}+D_{2.5}), \ldots,\ -(D_{n-0.5}+D_{n+0.5}))$

och med super- och subdiagonalkomponenterna $(D_{1.5},\ D_{2.5}, \ldots,\ D_{n-0.5})$.

Konstanten γ är $1/h^2$ och vektorn $\mathbf{b}(t)$ kommer att få första komponenten $D_{0.5}f_0(t)$ och sista komponenten $D_{n+0.5}f_L(t)$ (visa detta).

9.4 Olika fall av randvillkor

I PDE-problemen hittills har randvillkoren haft enkel form med kända funktionsuttryck för $u(0,t)$ och $u(L,t)$. Denna typ kallas för dirichletvillkor. Liksom vid randvärdesproblemen för ODE som behandlades i kapitel 7 kan randvillkoren innehålla derivatasamband och kallas då neumannvillkor (se avsnitt 7.5). Om väggen i vårt värmeledningsproblem är isolerad vid $x = L$

så blir randvillkoret $\frac{\partial u}{\partial x}(L,t) = 0$. I andra problem kan randvillkoret ha formen $\frac{\partial u}{\partial x}(L,t) = 0.1\,(u(L,t) - 17)$.

I dessa fall är temperaturen $u(x,t)$ okänd inte bara i det inre av väggen utan även på randen $x = L$, vilket medför att $u(L,t)$ måste inbegripas bland de obekanta. I algoritmen gäller nu $n = N$ med $x_n = L$.

Exemplet ovan med $\frac{\partial u}{\partial x}(L,t) = 0.1\,(u(L,t) - 17)$ hanteras på följande sätt. Approximera derivatan vid $x = x_n$ med centraldifferenskvot. Det ger $(u_{n+1} - u_{n-1})/2h = 0.1(u_n - 17)$, och vi får ett uttryck för den oönskade spökpunkten u_{n+1} som blir $u_{n+1} = u_{n-1} + 0.2hu_n - 3.4h$.
Den n-te ekvationen i differentialekvationssystemet (28) eller (34) som innehåller u_{n-1}, u_n och u_{n+1} måste vi ju alltid specialbehandla, och i den elimineras spökpunkten u_{n+1} genom insättning av randvillkorsapproximationen.

I matrisformuleringen (29) av differentialekvationssystemet påverkas sista matrisraden och sista komponenten i vektorn $\mathbf{b}$.

Om derivatavillkoret finns vid vänstra randen kommer i stället första raden i matrisen och första komponenten i $\mathbf{b}$ att påverkas.

9.5 Ickelinjära paraboliska PDE

Vi har hittills enbart behandlat *linjära* paraboliska partiella differentialekvationer. I värmeledningsproblemen har värmediffusiviteten antingen varit konstant som i ekvation (27) eller berott av x som i ekvation (33). Men diffusiviteten kan i vissa fall vara temperaturberoende, $D(u)$, vilket leder till en ickelinjär differentialekvation på formen $\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(D(u)\frac{\partial u}{\partial x}\right)$.

I vissa problem kan följande ickelinjära paraboliska PDE uppkomma:

$$\frac{\partial u}{\partial t} = D(u)\,\frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le L, \;\; t \ge 0 \tag{35}$$

Kända randvillkor av dirichlet- eller neumanntyp antas finnas vid $x = 0$ och $x = L$. Vid $t = 0$ känner vi temperaturfördelningen $u(x,0)$.

Den numeriska behandlingen inleds som tidigare med rumsdiskretisering enligt finitadifferensmetoden. Vi studerar differentialekvationen (35) i ett fall då $L = 1$ och randvillkoren är $u(0,t) = 0$ samt $u(1,t) = \cos 0.5t$. Begynnelsetemperaturen är $u(x,0) = x$, $0 \le x \le 1$.

För $D(u)$ antas följande samband finnas: $D(u) = \beta_0 + \beta_1 u$ med kända konstanter $\beta_0 = 0.03$ och $\beta_1 = 0.02$.

Rumsdiskretisering i N delintervall innebär vid dirichletvillkor att det finns $n = N-1$ okända temperaturfunktioner $u_i(t)$ vid $x_i = i\,h$ där $h = L/N$.

När högerledets andraderivata i (35) approximeras med differenskvot får vi differensekvationerna

$$\frac{du_i}{dt} = \frac{1}{h^2} D(u_i)(u_{i-1}(t) - 2u_i(t) + u_{i+1}(t)) \quad i = 1, 2, ..., n.$$

Det resulterar i följande system av ickelinjära differentialekvationer

$$\frac{d\mathbf{u}}{dt} = \frac{1}{h^2}\,\mathbf{F}(t, \mathbf{u}) \quad \text{med} \quad \mathbf{F}(t, \mathbf{u}) = \begin{pmatrix} D(u_1)\,(0 - 2u_1 + u_2) \\ D(u_2)\,(u_1 - 2u_2 + u_3) \\ D(u_3)\,(u_2 - 2u_3 + u_4) \\ \cdots \\ D(u_{n-1})\,(u_{n-2} - 2u_{n-1} + u_n) \\ D(u_n)\,(u_{n-1} - 2u_n + \cos 0.5t) \end{pmatrix} \tag{36}$$

Med en explicit algoritm som Eulers metod med formeln

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + \frac{\delta t}{h^2}\,\mathbf{F}(t^{(j)}, \mathbf{u}^{(j)})$$

spelar det ingen roll att problemet är ickelinjärt; beräkningsförfarandet är lika enkelt som vid linjära differentialekvationer. Stabilitetskravet hos Eulers metod gör att tidssteget måste vara litet för att inte oönskade oscillationer ska uppstå.

RK4 med noggrannhetsordning $\mathrm{O}((\delta t)^4)$ är en explicit metod som är smidig att använda. Tyvärr har den ju också en stabilitetströskel för tidssteget som det gäller att hålla sig under. Men om man är medveten om det, är RK4 ofta lämplig för ickelinjära paraboliska PDE.

Liksom tidigare är en implicit metod ett bra och säkert alternativ. Med Crank-Nicolsons metod slipper vi instabilitetsbekymret — den är ju stabil vilket tidssteg vi än tar. Vi kan enbart låta noggrannhetskravet styra valet av δt. Crank-Nicolsons metod, dvs trapetsmetoden, på differentialekvationssystemet (36) lyder

$$\mathbf{u}^{(j+1)} = \mathbf{u}^{(j)} + \frac{\delta t}{2h^2}\,\Big(\mathbf{F}(t^{(j)}, \mathbf{u}^{(j)}) + \mathbf{F}(t^{(j+1)}, \mathbf{u}^{(j+1)})\Big) \tag{37}$$

Inom varje tidssteg måste vi lösa ett ickelinjärt ekvationssystem för att beräkna $\mathbf{u}^{(j+1)}$. Snabbast konvergerande metod för detta är Newtons metod. Inför beteckningarna: $q = \delta t/2h^2$, $\mathbf{v} = \mathbf{u}^{(j+1)}$ och $\mathbf{w} = \mathbf{u}^{(j)} + q\,\mathbf{F}(t^{(j)}, \mathbf{u}^{(j)})$. Ekvation (37) kan därmed skrivas $\mathbf{v} - q\,\mathbf{F}(t^{(j+1)}, \mathbf{v}) - \mathbf{w} = \mathbf{0}$.

Det är ett ickelinjärt ekvationssystem på formen $\mathbf{f}(\mathbf{v}) = \mathbf{0}$ för bestämning av de okända temperaturvärdena $v_1, v_2, \ldots, v_n$ vid nästa tidssteg. För

att lösa detta med Newtons metod behövs jacobianen $\mathbf{J}$ till $\mathbf{f}$. Den erhålls som $\mathbf{J} = \mathbf{I} - q\,\mathbf{J}_F$ där $\mathbf{I}$ är enhetsmatrisen och $\mathbf{J}_F$ är jacobianmatrisen till $\mathbf{F}$. Elementen i $\mathbf{J}_F$ erhålls genom partiell derivering av komponenterna i högra uttrycket i (36) med avseende på u_1, u_2, osv. Matrisen blir tridiagonal (verifiera detta) med diagonalelementen $D'(u_i)(u_{i-1} - 2u_i + u_{i+1}) - 2D(u_i)$. Superdiagonalvektorn har komponenterna $D(u_1)$, $D(u_2), \ldots, D(u_{n-1})$ och subdiagonalvektorn har komponenterna $D(u_2)$, $D(u_3), \ldots, D(u_n)$.

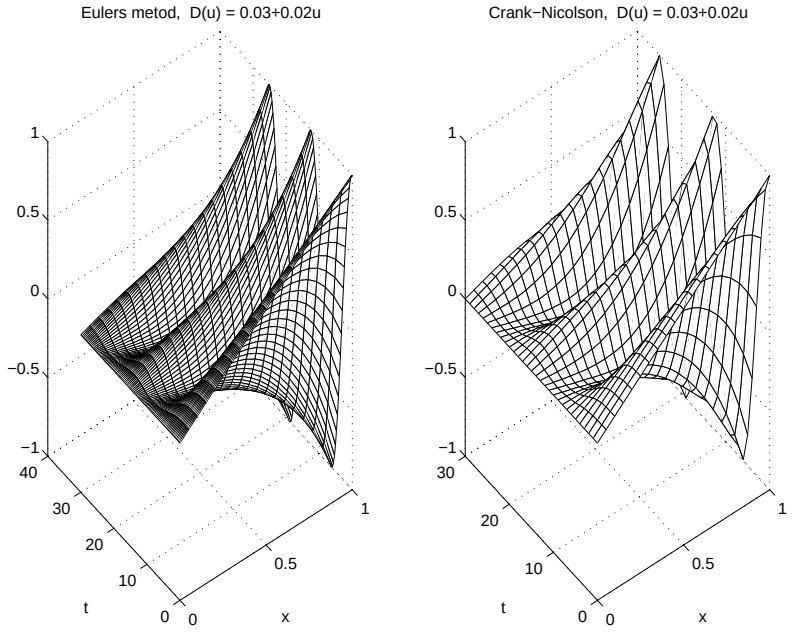
Vi skriver MATLAB-funktioner för uttrycket (36) och jacobianen $\mathbf{J}_F$:

```
function F=ftvarmol(t,u)
   global beta0 beta1
   n=length(u);  uslut=cos(0.5*t);
   D=beta0+beta1*u;
   F=D.*[[0;u(1:n-1)]-2*u+[u(2:n);uslut]];

function JF=ftvarmoljac(t,u)
   global beta0 beta1
   n=length(u); uslut=cos(0.5*t);
   D=beta0+beta1*u;
   Dprim=beta1*ones(n,1);
   g=[[0;u(1:n-1)]-2*u+[u(2:n);uslut]];
   dia=Dprim.*g-2*D;  sup=D(1:n-1);  sub=D(2:n);
   JF=diag(dia)+diag(sup,1)+diag(sub,-1);
```

Följande program löser den ickelinjära differentialekvationen (35). Rumsdiskretiseringen med FDM görs i 20 delintervall. För det omformade problemet (36) tillämpas först Eulers metod och därefter för jämförelsens skull Crank-Nicolsons metod.

```
  global beta0 beta1
  beta0=0.03;  beta1=0.02;
  N=20; h=1/N; n=N-1; x=h*(1:n)'; xx=[0;x;1];
  ustart=x;  tslut=30;
% Eulers metod, dt maximalt 0.03, instabil vid större dt
  dt=0.03;  q=dt/h^2;
  t=0; k=0;
  u=ustart;  uu=[0; u; 1];
  T=t; U=uu';
  while t<tslut-dt/2
    u=u+q*ftvarmol(t,u);
    t=t+dt; k=k+1;  % lagra vid vart tionde tidssteg
    if rem(k,10)==0, uu=[0; u; cos(0.5*t)]; T=[T; t]; U=[U; uu']; end
  end
  subplot(1,2,1), mesh(xx',T,U)
```

```
% Crank-Nicolsons metod
  dt=1;  q=dt/(2*h^2);
  t=0;  u=ustart;  uu=[0; u; 1]; T=t; U=uu';
  v=u;                          % gissning till lösningen vid nästa tidssteg
  while t<tslut-dt/2
    w=u+q*ftvarmol(t,u);  dvnorm=1; iter=0;                % Newtons metod
    while dvnorm>1e-5 & iter<10
      f=v-q*ftvarmol(t+dt,v)-w;
      J=eye(n)-q*ftvarmoljac(t+dt,v);
      dv=-J\f;  dvnorm=norm(dv,inf);
      v=v+dv; iter=iter+1;
    end
    if iter==10, disp('Ingen konvergens'), break, end
    u=v; t=t+dt; uu=[0; u; cos(0.5*t)];
    T=[T; t]; U=[U; uu'];
  end
  subplot(1,2,2), mesh(xx',T,U)
```



9.6 Värmeledningsproblem — ansatsmetod istället för FDM

Ansatsmetoderna *Galerkins metod* och *kollokation* som presenterades i avsnitt 7.9 och 7.10 som alternativ till finitadifferensmetoden för randvärdesproblem vid ordinära differentialekvationer kan även tillämpas vid partiella differentialekvationer. Här nöjer vi oss med att studera metoderna på paraboliska PDE i en rumsdimension.[7]

Låt oss pröva ansatsmetoderna på värmeledningsproblemet (27):

$$\frac{\partial u}{\partial t} = \beta \frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le L, \;\; 0 \le t \le t_{slut}$$

där β är känd konstant värmediffusivitet. Randvillkoren är $u(0,t) = f_0(t)$ och $u(L,t) = f_L(t)$. Dessutom finns ett begynnelsevillkor $u(x,0) = u_{start}(x)$.

9.6.1 Galerkins ansats – omformning från PDE- till ODE-problem

Galerkinansatsen för den sökta funktionen $u(x,t)$ utgörs av en linjärkombination av basfunktioner. Hattfunktionerna som utnyttjades i galerkinansatsen vid ODE är enklast och bekvämast att välja. Det innebär att lösningen till värmeledningsproblemet ansätts som

$$u(x,t) = \sum_{k=0}^{N} c_k(t)\phi_k(x), \quad \text{med} \;\; c_0(t) = f_0(t), \;\; c_N(t) = f_L(t).$$

Intervallet $0 \le x \le L$ delas i N delintervall och numreras från $x_0 = 0$ till $x_N = L$ med $h_i = x_{i+1} - x_i$. Hattfunktionerna $\phi_0(x)$, $\phi_1(x), \ldots, \phi_N(x)$ definieras av uttrycket (13) i avsnitt 7.9.1.

Ansatsen innebär att det finns $n = N-1$ tidsberoende koefficienter $c_k(t)$ att bestämma, och för detta behövs n ekvationer. Galerkinvillkoret att residualfunktionen ska vara ortogonal mot basfunktionerna leder till:

$$\int_0^L \left(\frac{\partial u}{\partial t} - \beta \frac{\partial^2 u}{\partial x^2} \right) \phi_i(x)\, dx = 0, \quad i = 1, 2, \ldots n. \tag{38}$$

Liksom tidigare utnyttjas partiell integration på termen med andraderivatan:

$$\int_0^L \frac{\partial^2 u}{\partial x^2}\, \phi_i(x)\, dx = \left[\frac{\partial u}{\partial x}\, \phi_i(x) \right]_0^L - \int_0^L \frac{\partial u}{\partial x}\, \phi_i'\, dx = 0 - \int_0^L \frac{\partial u}{\partial x}\, \phi_i'\, dx$$

[7] För PDE-problem i 2D och 3D hänvisas till kurser i finitaelementmetoden (FEM).

Integralsambandet (38) kan nu skrivas

$$\int_0^L \frac{\partial u}{\partial t}\,\phi_i(x)\,dx = -\beta \int_0^L \frac{\partial u}{\partial x}\,\phi_i'(x)\,dx.$$

Insättning av $u = \sum c_k(t)\phi_k(x)$ och dess derivator $\partial u/\partial x = \sum c_k(t)\phi_k'(x)$ och $\partial u/\partial t = \sum c_k'(t)\phi_k(x)$ samt vetskapen att hattfunktionsprodukter $\phi_k\phi_i$ är noll om $k < i-1$ eller $k > i+1$ ger

$$\sum_{k=i-1}^{i+1} c_k'(t) \int_0^L \phi_k\phi_i\,dx = -\beta \sum_{k=i-1}^{i+1} c_k(t) \int_0^L \phi_k'\phi_i'\,dx, \quad i = 1, 2, \ldots n.$$

Vi inför beteckningarna m_{ik} och s_{ik} för integralerna enligt:

$$m_{ik} = \int_0^L \phi_k\phi_i\,dx, \quad s_{ik} = -\int_0^L \phi_k'\phi_i'\,dx \tag{39}$$

Varje ekvation, $i = 1, 2, \ldots, n$, kommer därför att ha följande utseende:

$$m_{i,i-1}\frac{dc_{i-1}}{dt} + m_{ii}\frac{dc_i}{dt} + m_{i,i+1}\frac{dc_{i+1}}{dt} = \beta\,(s_{i,i-1}c_{i-1} + s_{ii}c_i + s_{i,i+1}c_{i+1}).$$

Ekvidistanta knutpunkter

Låt oss studera fallet med jämn indelning av $0 \le x \le L$ så att avståndet $h = L/N$ mellan knutpunkterna överallt är lika. Med integraluttrycken (39) och hattfunktionsuttrycket (13) erhålls:

$$m_{i,i-1} = \int_{x_{i-1}}^{x_i} \phi_{i-1}\phi_i dx = \frac{1}{h^2}\int_{x_{i-1}}^{x_i} (x_i - x)(x - x_{i-1})dx = \frac{h}{6}$$

$$m_{ii} = \int_{x_{i-1}}^{x_{i+1}} \phi_i^2 dx = \frac{1}{h^2}\int_{x_{i-1}}^{x_i} (x - x_{i-1})^2 dx + \frac{1}{h^2}\int_{x_i}^{x_{i+1}} (x_{i+1} - x)^2 dx = \frac{2h}{3}$$

$$m_{i,i+1} = \int_{x_i}^{x_{i+1}} \phi_{i+1}\phi_i dx = \frac{1}{h^2}\int_{x_i}^{x_{i+1}} (x - x_i)(x_{i+1} - x)dx = \frac{h}{6}$$

$$s_{i,i-1} = -\int_{x_{i-1}}^{x_i} \phi_{i-1}'\phi_i' dx = \int_{x_{i-1}}^{x_i} \frac{1}{h^2} dx = \frac{1}{h}$$

$$s_{ii} = -\int_{x_{i-1}}^{x_{i+1}} \phi_i'^2 dx = -(\frac{1}{h} + \frac{1}{h}) = -\frac{2}{h}$$

$$s_{i,i+1} = -\int_{x_i}^{x_{i+1}} \phi_i'\phi_{i+1}' dx = \frac{1}{h}$$

Differentialekvationssambanden blir i detta fall

$$\frac{h}{6}\left(\frac{dc_{i-1}}{dt}+4\frac{dc_i}{dt}+\frac{dc_{i+1}}{dt}\right)=\frac{\beta}{h}\left(c_{i-1}-2c_i+c_{i+1}\right),\quad i=1,2,\ldots,n. \quad (40)$$

Första och sista ekvationen i (40) är speciella eftersom de innehåller kända randvillkor. I ekvationen för $i=1$ ingår randfunktionen $c_0(t)=f_0(t)$ och dess derivata $c_0'(t)=f_0'(t)$ som ju också är känd. Första ekvationen blir

$$\frac{h}{6}\left(4\frac{dc_1}{dt}+\frac{dc_2}{dt}\right)=\frac{\beta}{h}\left(-2c_1+c_2\right)+\frac{\beta}{h}f_0(t)-\frac{h}{6}f_0'(t).$$

I sista ekvationen när $i=n=N-1$, ingår den bekanta randfunktionen $c_N(t)=c_{n+1}(t)=f_L(t)$ och $c_N'(t)=f_L'(t)$, vilket leder till

$$\frac{h}{6}\left(\frac{dc_{n-1}}{dt}+4\frac{dc_n}{dt}\right)=\frac{\beta}{h}\left(c_{n-1}-2c_n\right)+\frac{\beta}{h}f_L(t)-\frac{h}{6}f_L'(t).$$

De n ekvationerna kan samlas i matris-vektor-form:

$$\mathbf{M}\frac{d\mathbf{c}}{dt}=\beta\,\mathbf{S}\mathbf{c}+\mathbf{b}(t),\quad \mathbf{c}(0)=\mathbf{c}_{start} \quad (41)$$

där $\mathbf{M}$ och $\mathbf{S}$ är de tridiagonala matriserna

$$\mathbf{M}=\frac{h}{6}\begin{pmatrix}4&1&0&0&\cdots\\1&4&1&0&\cdots\\0&1&4&1&\cdots\\\vdots&\vdots&&\ddots&\\0&0&\cdots&1&4\end{pmatrix},\quad \mathbf{S}=\frac{1}{h}\begin{pmatrix}-2&1&0&0&\cdots\\1&-2&1&0&\cdots\\0&1&-2&1&\cdots\\\vdots&\vdots&&\ddots&\\0&0&\cdots&1&-2\end{pmatrix}$$

I vektorn $\mathbf{b}(t)$ är elementen $b_2,\ldots,b_{n-1}$ noll, medan

$$b_1(t)=\frac{\beta}{h}f_0(t)-\frac{h}{6}f_0'(t),\quad b_n(t)=\frac{\beta}{h}f_L(t)-\frac{h}{6}f_L'(t).$$

Med Galerkins ansatsmetod har alltså värmeledningsproblemet omformats till systemet (41) av ordinära differentialekvationer.

Det behövs värden vid $t=0$ till vektorn $\mathbf{c}_{start}$ i (41). Startvärdena hämtas ur begynnelsevillkoret $u(x,0)=u_{start}(x)$ till värmeledningsproblemet. Eftersom galerkinansatsen är en linjärkombination av hattfunktioner (13) så gäller vid knutpunkterna x_i följande enkla samband (vid tiden t):

$$u(x_i,t)=\sum_{k=0}^{N}c_k(t)\phi_k(x_i)=c_i(t),\quad i=0,1,\ldots,N.$$

Vid $t=0$ har vi alltså $c_i(0)=u(x_i,0)=u_{start}(x_i)$. Vektorn $\mathbf{c}_{start}$ består av de n komponenterna $c_1(0)$, $c_2(0),\ldots,c_n(0)$.

9.6.2 Implicita metoder för lösning av ODE-systemet

Trots att Galerkins metod och finitadifferensmetoden har helt olika angreppssätt på värmeledningsproblemet, leder bägge ansatserna fram till ett system av första ordningens ordinära differentialekvationer — ekvation (41) med galerkinansatsen och ekvation (29) i avsnitt 9.2.2 med FDM. Strukturen på ODE-systemen är olika, men båda har egenskapen att vara styva differentialekvationsproblem. För sådana gäller av stabilitetsskäl att en implicit numerisk lösningsmetod såsom bakåteulermetoden eller trapetsmetoden är att föredra framför explicita stegmetoder av typen Eulers metod och rungekuttametoder (RK4, ode45, m fl).

Låt δt beteckna tidssteget. Bakåteulermetoden applicerad på differentialekvationssystemet (41) ger

$$\mathbf{M}\,(\mathbf{c}^{(j+1)} - \mathbf{c}^{(j)}) = \delta t\,(\beta\,\mathbf{S}\mathbf{c}^{(j+1)} + \mathbf{b}(t^{(j+1)}))$$

som efter lite omformning blir

$$(\mathbf{M} - \beta\,\delta t\,\mathbf{S})\,\mathbf{c}^{(j+1)} = \mathbf{M}\mathbf{c}^{(j)} + \delta t\,\mathbf{b}(t^{(j+1)}) \tag{42}$$

I varje tidssteg gäller det att lösa ett linjärt ekvationssystem med den tridiagonala systemmatrisen $\mathbf{M} - \beta\,\delta t\,\mathbf{S}$.

Metoden är stabil för varje val av tidssteget δt. Noggrannhetsmässigt är den inte så bra eftersom trunkeringsfelet är $O(\delta t)$. Bättre ur noggrannhetssynpunkt är trapetsmetoden vars trunkeringsfel är proportionellt mot $(\delta t)^2$.

Trapetsmetoden tillämpad på ODE-systemet (41) ger

$$\mathbf{M}\,(\mathbf{c}^{(j+1)} - \mathbf{c}^{(j)}) = \frac{\delta t}{2}\,(\beta\,\mathbf{S}\mathbf{c}^{(j)} + \mathbf{b}(t^{(j)}) + \beta\,\mathbf{S}\mathbf{c}^{(j+1)} + \mathbf{b}(t^{(j+1)}))$$

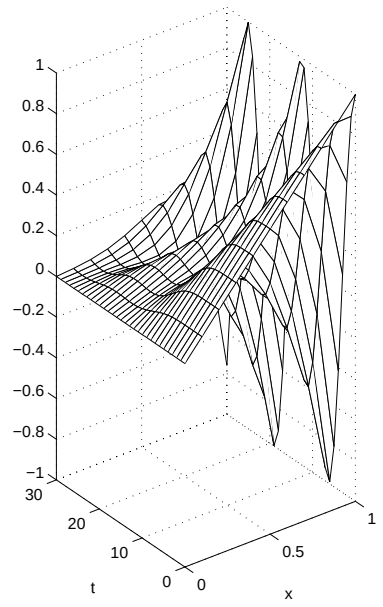
Efter omflyttning så att $\mathbf{c}^{(j+1)}$-termer hamnar på vänster sida och termer med $\mathbf{c}^{(j)}$ på höger sida erhålls

$$(\mathbf{M} - \frac{\beta\,\delta t}{2}\mathbf{S})\,\mathbf{c}^{(j+1)} = (\mathbf{M} + \frac{\beta\,\delta t}{2}\mathbf{S})\,\mathbf{c}^{(j)} + \frac{\delta t}{2}\,(\mathbf{b}(t^{(j)}) + \mathbf{b}(t^{(j+1)})) \tag{43}$$

Detta är ett linjärt system med tridiagonal systemmatris $\mathbf{M} - \frac{\beta\,\delta t}{2}\mathbf{S}$ för beräkning av $\mathbf{c}^{(j+1)}$ vid tiden $t_j + \delta t$ då koefficientvektorn vid tidpunkten t_j är känd. Lösning av systemet (43) måste därför göras vid varje tidssteg.

Exempel: Lös värmeledningsproblemet i ett fall med $\beta = 0.01$ och $L = 1$. Randvillkor: $u(0,t) = 0$, $u(1,t) = \cos 0.5t$. Begynnelsevillkor: $u(x,0) = x$, $0 \le x \le 1$. Lös fram till $t_{slut} = 30$. Med Galerkins metod och trapetsmetoden med tidssteget $\delta t = 1$ får vi följande algoritm:

```
% varmgalerk, du/dt = beta*d2u/dx2
  beta=0.01;
  N=10;  h=1/N; n=N-1;
  x=h*(1:n)'; xx=[0; x; 1];
  Mdia=2*h/3*ones(n,1);
  Msup=h/6*ones(n-1,1);
  M=diag(Mdia)+diag(Msup,1)+diag(Msup,-1);
  Sdia=-2/h*ones(n,1);
  Ssup=1/h*ones(n-1,1);
  S=diag(Sdia)+diag(Ssup,1)+diag(Ssup,-1);

  t=0; T=t; u0=x; uu=[0; u0; 1]; U=uu';
  c=u0;  tslut=30;
  dt=1; q=beta*dt/2;
  Cdia=Mdia-q*Sdia;
  Csup=Msup-q*Ssup;
  bn=beta/h*cos(0.5*t)+0.5*h/6*sin(0.5*t);
  while t<tslut-dt/2
    t=t+dt;  bnp=beta/h*cos(0.5*t)+0.5*h/6*sin(0.5*t);
    b=[zeros(n-1,1); bn+bnp];
    HL=(M+q*S)*c+dt/2*b;
    c=tridia(Cdia,Csup,Csup,HL);
    uu=[0; c; cos(0.5*t)];  U=[U; uu']; T=[T; t];
    bn=bnp;
  end
  mesh(xx',T,U)
```



9.6.3 Kollokation – omformning från PDE- till ODE-problem

I värmeledningsproblemet

$$\frac{\partial u}{\partial t} = \beta \frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le L, \quad u(0,t) = f_0(t),\ u(L,t) = f_L(t),$$

approximerar vi $u(x,t)$ med en linjärkombination av väl valda basfunktioner. I kollokationsansatsen är bellsplines lämpliga basfunktioner enligt:

$$u(x,t) \approx \sum_{k=0}^{n+1} c_k(t) B_k(x).$$

I värmeledningsekvationen ersätts tidsderivatan $\partial u/\partial t$ av $\sum_{k=0}^{n+1} c_k'(t) B_k(x)$ och andraderivatan i rummet av $\sum_{k=0}^{n+1} c_k(t) B_k''(x)$.

Intervallet $0 \le x \le L$ antas ha samma ekvidistanta knutpunktsindelning och numrering som i galerkinansatsen, alltså $N = n+1$ lika delintervall med $x_0 = 0$, $x_N = x_{n+1} = L$ och $h = L/N$.

För att kunna bestämma koefficienterna $c_0(t)$, $c_1(t), \ldots, c_{n+1}(t)$ måste vi ha tillgång till $n+2$ ekvationer. Interpolation i de n inre knutpunkterna ger oss n samband; de övriga erhålls från randvillkoren.

Interpolationskravet innebär att differentialekvationen $\frac{\partial u}{\partial t} = \beta \frac{\partial^2 u}{\partial x^2}$ ska vara satisfierad i knutpunkterna $x_1, x_2, \ldots, x_n$. Det leder till

$$\sum_{k=0}^{n+1} c_k'(t) B_k(x_i) = \beta \sum_{k=0}^{n+1} c_k(t) B_k''(x_i), \quad i = 1, 2, \ldots, n.$$

Vid varje knutpunkt x_i är bara tre bellsplinekurvor skilda från noll och deras värden finns tabellerade i (24). Summorna i vänster- och högerledet består därför av tre termer:

$$\text{VL} = \frac{dc_{i-1}}{dt} B_{i-1}(x_i) + \frac{dc_i}{dt} B_i(x_i) + \frac{dc_{i+1}}{dt} B_{i+1}(x_i),$$
$$\text{HL} = \beta \left(c_{i-1} B_{i-1}''(x_i) + c_i B_i''(x_i) + c_{i+1} B_{i+1}''(x_i) \right),$$

vilket leder till följande differentialekvationssamband

$$\frac{1}{6}\left(\frac{dc_{i-1}}{dt} + 4\frac{dc_i}{dt} + \frac{dc_{i+1}}{dt} \right) = \frac{\beta}{h^2}\left(c_{i-1} - 2c_i + c_{i+1} \right), \quad i = 1, 2, \ldots, n.$$

Om båda sidorna multipliceras med h blir uttrycket identiskt med (40) som erhölls med Galerkins ansats:

$$\frac{h}{6}\left(\frac{dc_{i-1}}{dt} + 4\frac{dc_i}{dt} + \frac{dc_{i+1}}{dt} \right) = \frac{\beta}{h}\left(c_{i-1} - 2c_i + c_{i+1} \right), \quad i = 1, 2, \ldots, n. \quad (44)$$

Randvillkoret $u(x_0,t) = f_0(t)$ approximeras av

$$c_{-1}B_{-1}(x_0) + c_0B_0(x_0) + c_1B_1(x_0) = f_0(t) \implies \frac{1}{6}(c_{-1} + 4c_0 + c_1) = f_0(t).$$

Här finns spökkoefficienten $c_{-1} = 6f_0(t) - 4c_0 - c_1$ som vi gärna vill bli av med. Vi deriverar randvillkoret vid x_0: $\frac{\partial u}{\partial t}(x_0,t) = f_0'(t)$ och utnyttjar att $\frac{\partial u}{\partial t} = \beta\,\frac{\partial^2 u}{\partial x^2}$ ska gälla vid vänstra randen, alltså $f_0'(t) = \frac{\beta}{h^2}(c_{-1} - 2c_0 + c_1)$. Vi stoppar in uttrycket för c_{-1}: $f_0'(t) = \frac{\beta}{h^2}(6f_0(t) - 4c_0 - c_1 - 2c_0 + c_1)$. Ur detta följer direkt ett uttryck för koefficienten c_0:

$$c_0(t) = f_0(t) - \frac{h^2}{6\beta}\,f_0'(t).$$

Vid randen $x = L$ där randvillkoret lyder $u(x_{n+1},t) = f_L(t)$ förfar man på motsvarande sätt. Spökkoefficienten $c_{n+2} = 6f_L(t) - 4c_{n+1} - c_n$ elimineras och man får ett uttryck för koefficienten c_{n+1}:

$$c_{n+1}(t) = f_L(t) - \frac{h^2}{6\beta}\,f_L'(t).$$

Vi återvänder till sambanden (44). I första ekvationen flyttas den kända derivatan $c_0'(t)$ över till höger sida:

$$\frac{h}{6}\left(4\frac{dc_1}{dt} + \frac{dc_2}{dt}\right) = \frac{\beta}{h}\,(-2c_1 + c_2) + \frac{\beta}{h}c_0(t) - \frac{h}{6}c_0'(t).$$

I den n-te ekvationen flyttas $c_{n+1}'(t)$ över till högersidan:

$$\frac{h}{6}\left(\frac{dc_{n-1}}{dt} + 4\frac{dc_n}{dt}\right) = \frac{\beta}{h}\,(c_{n-1} - 2c_n) + \frac{\beta}{h}c_{n+1}(t) - \frac{h}{6}c_{n+1}'(t).$$

Det innebär att ekvationerna i (44) kan skrivas på matris-vektor-form

$$\mathbf{M}\frac{d\mathbf{c}}{dt} = \beta\,\mathbf{S}\mathbf{c} + \mathbf{b}(t), \quad \mathbf{c}(0) = \mathbf{c}_{start} \tag{45}$$

där $\mathbf{M}$ och $\mathbf{S}$ är samma tridiagonala matriser som tidigare:

$$\mathbf{M} = \frac{h}{6}\begin{pmatrix} 4 & 1 & 0 & 0 & \cdots \\ 1 & 4 & 1 & 0 & \cdots \\ 0 & 1 & 4 & 1 & \cdots \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & \cdots & 1 & 4 \end{pmatrix}, \quad \mathbf{S} = \frac{1}{h}\begin{pmatrix} -2 & 1 & 0 & 0 & \cdots \\ 1 & -2 & 1 & 0 & \cdots \\ 0 & 1 & -2 & 1 & \cdots \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & \cdots & 1 & -2 \end{pmatrix}$$

I vektorn $\mathbf{b}(t)$ gäller att elementen $b_2, \ldots, b_{n-1}$ är noll medan

$$b_1(t) = \frac{\beta}{h}c_0(t) - \frac{h}{6}c_0'(t) = \frac{\beta}{h}f_0(t) - \frac{h}{3}f_0'(t) + \frac{h^3}{36\beta}\,f_0''(t),$$

$$b_n(t) = \frac{\beta}{h}c_{n+1}(t) - \frac{h}{6}c_{n+1}'(t) = \frac{\beta}{h}f_L(t) - \frac{h}{3}f_L'(t) + \frac{h^3}{36\beta}\,f_L''(t).$$

Startvektorn $\mathbf{c}_{start}$ behövs; den erhålls genom att värmeledningsproblemets givna begynnelsefunktion $u_{start}(x)$ interpoleras med en linjärkombination av bellsplines i knutpunkterna:

$$u_{start}(x_i) = \sum_{k=i-1}^{i+1} c_k(0)B_k(x_i) = \frac{1}{6}(c_{i-1}(0)+4c_i(0)+c_{i+1}(0)),\ i = 1,\, 2, \ldots, n.$$

Det leder till ett tridiagonalt system för bestämning av vektorn $\mathbf{c}_{start}$:

$$\frac{1}{6}\begin{pmatrix} 4 & 1 & 0 & 0 & \cdots \\ 1 & 4 & 1 & 0 & \cdots \\ 0 & 1 & 4 & 1 & \cdots \\ \vdots & \vdots & & \ddots & \\ 0 & 0 & \cdots & 1 & 4 \end{pmatrix}\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} u_{start}(x_1) - c_0(0)/6 \\ u_{start}(x_2) \\ u_{start}(x_3) \\ \vdots \\ u_{start}(x_n) - c_{n+1}(0)/6 \end{pmatrix}$$

Beteckna högerledet med $\mathbf{v}$. I vänsterledet känner vi igen matrisen $\mathbf{M}$ så när som på faktorn h. Lösningen till ekvationssystemet $\mathbf{M}\mathbf{c}_{start} = h\,\mathbf{v}$ ger alltså startvektorn till ODE-systemet (45), som i sin tur löses med bakåteulermetoden enligt (42) eller trapetsmetoden (43).

När man i varje tidssteg har beräknat vektorn $\mathbf{c}$ med komponenterna $c_1(t), \ldots, c_n(t)$, bestäms $u(x,t)$ av en känd linjärkombination av bellsplines. I intervallets inre knutpunkter $x_1, \ldots, x_n$ gäller

$$u(x_i,t) = \sum_{k=i-1}^{i+1} c_k(t)B_k(x_i) = \frac{1}{6}(c_{i-1}(t) + 4c_i(t) + c_{i+1}(t)),\ \ i = 1,\, 2, \ldots, n.$$

Med matrisformulering kan temperaturvektorn $\mathbf{u}(t)$ skrivas

$$\mathbf{u} = \frac{1}{h}\mathbf{M}\mathbf{c} + \frac{1}{6}\begin{pmatrix} c_0(t) \\ 0 \\ \vdots \\ c_{n+1}(t) \end{pmatrix}$$

Därmed kan algoritmen anses vara klar och problemet löst.

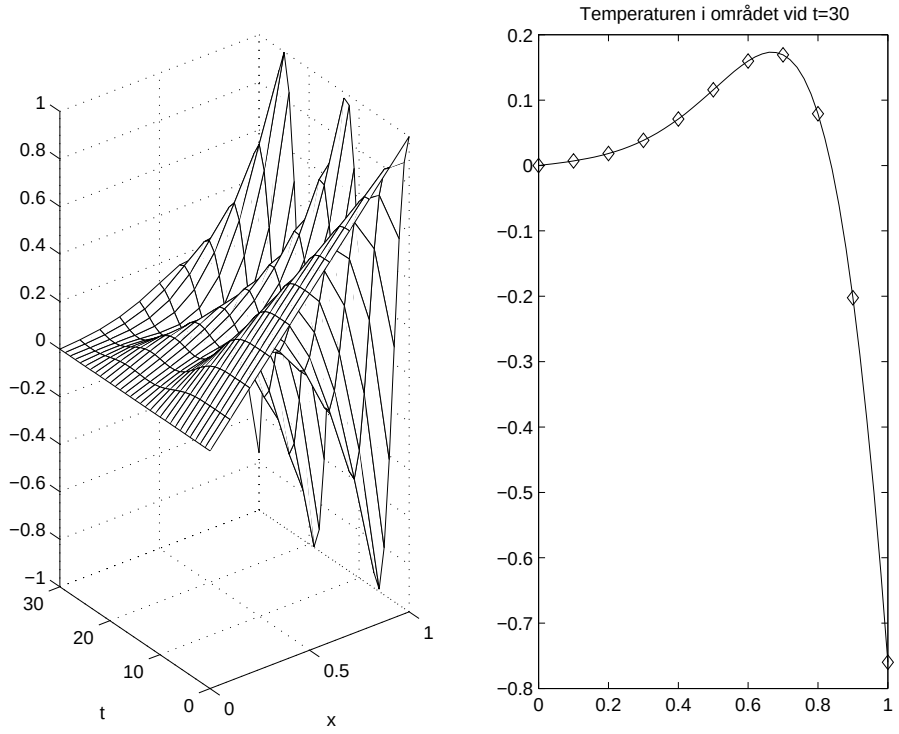
Vi kan snygga till temperaturkurvorna mer, eftersom en linjärkombination av bellsplines innebär att en mjuk kurvform kan åstadkommas. Spökkoefficienterna c_{-1} och c_{n+2} behöver plockas fram igen för att det ska vara möjligt att rita upp temperaturkurvan i hela intervallet från $x_0 = 0$ till $x_{n+1} = L$. Antalet knutpunkter är faktiskt $n + 2$, vilket innebär att $n + 4$ bellsplinekurvor ger positiva bidrag i $0 \leq x \leq L$.

Totalt sett gäller $u(x,t) = \sum_{k=-1}^{n+2} c_k(t)B_k(x)$ där vi förutom koefficienterna $c_0(t)$, $c_1(t), \ldots, c_{n+1}(t)$ behöver utnyttja spökkoefficienterna $c_{-1}(t) = 6f_0(t) - 4c_0(t) - c_1(t)$ och $c_{n+2}(t) = 6f_L(t) - 4c_{n+1}(t) - c_n(t)$.

Exempel: Pröva algoritmen med kollokation och trapetsmetoden på samma värmeledningsproblem som i galerkinexemplet. Data: $\beta = 0.01$, $L = 1$. Randvillkor: $u(0,t) = 0$, $u(1,t) = \cos 0.5t$. Begynnelsevillkor: $u(x,0) = x$, $0 \leq x \leq 1$. Lös med tidssteget $\delta t = 1$ fram till $t_{slut} = 30$.

```
% varmkollok
  beta=0.01;
  N=10;  h=1/N;  n=N-1;
  x=h*(1:n)'; xx=[0; x; 1];
  Mdia=2*h/3*ones(n,1);
  Msup=h/6*ones(n-1,1);
  M=diag(Mdia)+diag(Msup,1)+diag(Msup,-1);
  Sdia=-2/h*ones(n,1);
  Ssup=1/h*ones(n-1,1);
  S=diag(Sdia)+diag(Ssup,1)+diag(Ssup,-1);

  t=0; T=t; u0=x; uu=[0; u0; 1]; U=uu';
  v=u0; v(n)=v(n)-1/6; v=h*v;
  c=tridia(Mdia,Msup,Msup,v);

  tslut=30;
  dt=1; q=beta*dt/2;
  Cdia=Mdia-q*Sdia;
  Csup=Msup-q*Ssup;
  bn=beta/h*cos(0.5*t)+0.5*h/3*sin(0.5*t)+...
     h^3/(36*beta)*0.25*cos(0.5*t);

  while t<tslut-dt/2
    t=t+dt;
    bnp=beta/h*cos(0.5*t)+0.5*h/3*sin(0.5*t)+...
        h^3/(36*beta)*0.25*cos(0.5*t);
    b=[zeros(n-1,1); bn+bnp];
    HL=(M+q*S)*c+dt/2*b;
    c=tridia(Cdia,Csup,Csup,HL);
```

```
    c_nplus1=cos(0.5*t)+0.5*h^2/(6*beta)*sin(0.5*t);
    u=M*c/h+[zeros(n-1,1); c_nplus1/6];
    uu=[0; u; cos(0.5*t)];
    U=[U; uu']; T=[T; t];
    bn=bnp;
  end
  subplot(1,2,1), mesh(xx',T,U), xlabel('x'), ylabel('t')

% Använd info om koeff och bellsplines för att rita mjuk tempkurva
  c_spoek=6*cos(0.5*t)-4*c_nplus1-c(n);
  C=[-c(1); 0; c; c_nplus1; c_spoek];
  X=(0:0.02:1)';
  Us=0;
  for k=1:n+4, Us=Us+C(k)*fbelleq(k-1,X,0,1,n+2); end
  subplot(1,2,2), plot(xx,uu,'d', X,Us)
  title('Temperaturen i området vid t=30')
```



9.7 Parabolisk PDE i två rumsvariabler

Temperaturvariationen $u(x, y, t)$ i ett block med rektangulärt tvärsnitt och värmediffusivitet β beskrivs av värmeledningsekvationen i två rumsvariabler,

$$\frac{\partial u}{\partial t} = \beta \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \quad 0 \le x \le L_x, \;\; 0 \le y \le L_y, \;\; 0 \le t \le t_{slut} \qquad (46)$$

Blocket har känd temperatur, låt oss säga 100^oC, då det vid tiden $t = 0$ placeras med två blocksidor i en vägghörna med konstant tiogradig temperatur: $u(0, y, t) = 10, \; u(x, 0, t) = 10$. Vid de båda andra blocksidorna gäller rumstemperaturen 20^oC, alltså $u(L_x, y, t) = 20, \; u(x, L_y, t) = 20$.

Vi använder finitadifferensmetoden för rumsdiskretiseringen med N_x delintervall i x-led och N_y i y-led. $h_x = L_x/N_x$ och $h_y = L_y/N_y$. Eftersom temperaturen är känd överallt på randen gäller att antalet obekanta i x- och y-led är $n_x = N_x - 1$ och $n_y = N_y - 1$.

Låt $u_{ij}(t)$ vara approximation till $u(x_j, y_i, t)$. Matrisen med obekanta temperaturfunktioner kommer att få n_y rader och n_x kolumner.

I vårt exempel är $L_x = 0.4, \; L_y = 0.8$. Med $N_x = 4, \; N_y = 8$ blir steget $h_x = h_y = 0.1$.

Det blir sju gånger tre obekanta temperaturvärden att beräkna i varje tidssteg.

Med differenskvot enligt högerledet i formel (28) för andraderivatorna erhålls differentialekvationssystemet

10	10	10	10	20
10	u_{11}	u_{12}	u_{13}	20
10	u_{21}	u_{22}	u_{23}	20
10	u_{31}	u_{32}	u_{33}	20
10	u_{41}	u_{42}	u_{43}	20
10	u_{51}	u_{52}	u_{53}	20
10	u_{61}	u_{62}	u_{63}	20
10	u_{71}	u_{72}	u_{73}	20
10	20	20	20	20

$$\frac{du_{ij}}{dt} = \beta \left(\frac{u_{i,j+1}(t) - 2u_{ij}(t) + u_{i,j-1}(t)}{h_x^2} + \frac{u_{i+1,j}(t) - 2u_{ij}(t) + u_{i-1,j}(t)}{h_y^2} \right) \qquad (47)$$

När samma rumssteg används i x- och y-led, $h_x = h_y = h$, övergår (47) till

$$\frac{du_{ij}}{dt} = \frac{\beta}{h^2} \left(u_{i+1,j}(t) + u_{i-1,j}(t) + u_{i,j+1}(t) + u_{i,j-1}(t) - 4u_{ij} \right). \qquad (48)$$

Den enklaste metoden för numerisk behandling av (48) är Eulers metod. Instabilitetsegenskapen gör den dock praktiskt oanvändbar vid större PDE-problem, eftersom pyttesmå tidssteg krävs för att inte beräkningarna ska spåra ur.

I vårt aktuella exempel visar det sig att tidssteget $\delta t = 0.4$ leder till instabil lösning, medan $\delta t = 0.2$ ger stabil lösning. Det kan testas med följande lilla eulerprogram:

```
beta=0.01;  h=0.1;
dt=input('Tidssteg: ');
q=beta*dt/h^2;
t=0;
u=100*ones(7,3);
U=[10*ones(9,1) [10 10 10; u; 20 20 20] 20*ones(9,1)]
for stegnr=1:6                    % U beräknas under 6 tidssteg
  V=U;
  for j=2:4
    for i=2:8
      V(i,j)=U(i,j)+q*(U(i+1,j)+U(i-1,j)+U(i,j+1)+U(i,j-1)-4*U(i,j));
    end
   end
  t=t+dt; U=V
end
```

Crank-Nicolsons metod (32) som vi använde vid en rumsvariabel, kan utvidgas till 2D-fallet, och den är stabil för alla val av tidssteg. Vi återkommer till denna utmärkta metod i avsnitt 10.6, efter den numeriska hanteringen av elliptiska PDE-problem.

9.7.1 ADI-metoden

Här ska vi presentera en metod som kallas ADI-metoden, där ADI står för *Alternating Direction Implicit Method*. Den kan användas för paraboliska PDE i specialfallet när området i rummet är rektangulärt. ADI-metoden innebär att man turas om att räkna på de obekanta funktionerna $u_{i,j}(t)$ vektorvis i y-led och x-led.

Varje tidssteg omfattar två beräkningssteg, först ett halvt tidssteg med Eulers metod i x-led och den implicita bakåteuler (31) i y-riktningen (i vårt exempel tre kolumnvektorer att beräkna). Därefter utförs ett halvt tidssteg tvärtom, det vill säga med Eulers metod i y-led och bakåteuler i x-led (sju radvektorer att beräkna). Man kan visa att detta förfarande är stabilt för alla val av tidssteget δt.

Vi inför beteckningen $v_{ij}(t)$ för $u(x_j, y_i, t + \frac{\delta t}{2})$ och $q_x = \beta\,\delta t/2h_x^2$ och $q_y = \beta\,\delta t/2h_y^2$.

Formeln för första halva tidssteget kan skrivas

$$v_{ij} = u_{ij} + q_x(u_{i,j-1} - 2u_{ij} + u_{i,j+1}) + \\ + q_y(v_{i-1,j} - 2v_{ij} + v_{i+1,j}).$$

Med de obekanta v-värdena överflyttade till vänster sida får vi ett ekvationssystem att lösa för var och en av de tre kolumnerna $j = 1, 2, 3$.

10	10	10	10	20
10	v_{11}	v_{12}	v_{13}	20
10	v_{21}	v_{22}	v_{23}	20
10	v_{31}	v_{32}	v_{33}	20
10	v_{41}	v_{42}	v_{43}	20
10	v_{51}	v_{52}	v_{53}	20
10	v_{61}	v_{62}	v_{63}	20
10	v_{71}	v_{72}	v_{73}	20
10	20	20	20	20

$$\begin{cases} v_{1j} - q_y(-2v_{1j} + v_{2j}) & = u_{1j} + q_y 10 & +q_x(u_{1,j-1} - 2u_{1j} + u_{1,j+1}) \\ v_{2j} - q_y(v_{1j} - 2v_{2j} + v_{3j}) & = u_{2j} & +q_x(u_{2,j-1} - 2u_{2j} + u_{2,j+1}) \\ v_{3j} - q_y(v_{2j} - 2v_{3j} + v_{4j}) & = u_{3j} & +q_x(u_{3,j-1} - 2u_{3j} + u_{3,j+1}) \\ \cdots & = & \cdots \\ v_{7j} - q_y(v_{6j} - 2v_{7j}) & = u_{7j} + q_y 20 & +q_x(u_{7,j-1} - 2u_{7j} + u_{7,j+1}) \end{cases}$$

Det kan skrivas i matrisform:

$$(\mathbf{I} - q_y \mathbf{A}_y)\mathbf{v}_j = \mathbf{u}_j + q_y \mathbf{b}_y + q_x \left(\mathbf{u}_{j-1} - 2\mathbf{u}_j + \mathbf{u}_{j+1}\right)$$

där $\mathbf{A}_y$ är tridiagonal matris med sju rader och kolumner, och vektorn med index j betyder j-te kolumnen i matrisen av temperaturvärden. Vektorn $\mathbf{b}_y$ är som tidigare en randvärdesvektor.

När ekvationssystemen har lösts för alla j-värden är temperaturmatrisen klar vid tiden $t + \frac{\delta t}{2}$. Man alternerar riktning, det vill säga använder Eulers metod i y-led och bakåteuler i x-led. Alla v_{ij} är kända och vi låter u_{ij} vara okända värden ett halvt tidssteg framåt. Formeln blir:

$$u_{ij} = v_{ij} + q_x(u_{i,j-1} - 2u_{ij} + u_{i,j+1}) + q_y(v_{i-1,j} - 2v_{ij} + v_{i+1,j}).$$

Obekanta u-värden måste flyttas över till vänster sida. I vårt fall då $n_x = 3$ får vi följande system av tre ekvationer som ska lösas för var och en av de sju raderna ($i = 1, 2, ..., 7$) i temperaturmatrisen.

$$\begin{cases} u_{i1} - q_x(-2u_{i1} + u_{i2}) & = v_{i1} + q_x 10 & +q_y(v_{i-1,1} - 2v_{i1} + v_{i+1,1}) \\ u_{i2} - q_x(u_{i1} - 2u_{i2} + u_{i3}) & = v_{i2} & +q_y(v_{i-1,2} - 2v_{i2} + v_{i+1,2}) \\ u_{i3} - q_x(u_{i2} - 2u_{i3}) & = v_{i3} + q_x 20 & +q_y(v_{i-1,3} - 2v_{i3} + v_{i+1,3}) \end{cases}$$

Det kan skrivas i matrisform

$$(\mathbf{I} - q_x \mathbf{A}_x)\mathbf{u}_i = \mathbf{v}_i + q_x \mathbf{b}_x + q_y \left(\mathbf{v}_{i-1} - 2\mathbf{v}_i + \mathbf{v}_{i+1}\right)$$

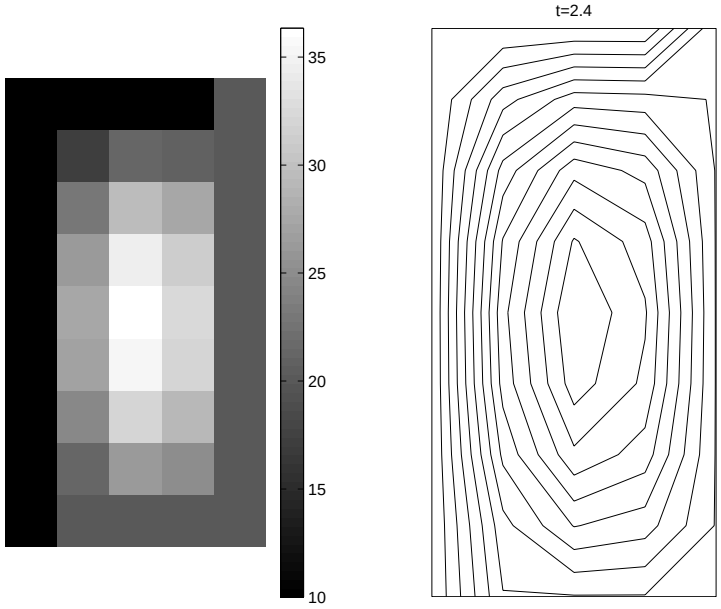
där $\mathbf{A}_x$ i vårt exempel är en 3×3-matris. Radvektorerna som ingår måste transponeras till kolumnvektorer innan ekvationssystemen kan lösas.

Vi visar algoritmen i form av ett MATLAB-program. Värmediffusiviteten i blocket antas vara $\beta = 0.01$. Temperaturen som vid starten är hundra grader sjunker successivt eftersom ingen värme tillförs. Den utjämnas slutligen till en stationär fördelning då alla temperaturvärden i blocket ligger mellan randtemperaturerna tio och tjugo grader.

Låt oss lösa avsvalningsproblemet från 100^oC vid $t = 0$ till en tidpunkt då ingen punkt i blocket är varmare än kroppstemperatur, alltså 37^oC.

ADI-metoden är stabil vilket tidssteg man än väljer, men för att få god noggrannhet i resultatet måste δt hållas ganska litet. I programmet nedan använder vi oss av $\delta t = 0.4$.

En exekvering av programmet nedan ger att blockets maxtemperatur sjunkit under 37^o vid tiden $t = 2.4$. Med `imagesc(Utot)` visas temperaturen i blocket i olika färgnyanser (tyvärr endast gråskala här).




```
% varm2Dblock, värmeledning 2D med ADI-metoden
  beta=0.01;
  Lx=0.4;  Ly=0.8;
  Nx=4;  nx=Nx-1; hx=Lx/Nx,
  x=0:hx:Lx;
  Ny=8;  ny=Ny-1; hy=Ly/Ny,
  y=-(0:hy:Ly);
  [X,Y]=meshgrid(x,y);
  Ax=-2*eye(nx)+diag(ones(nx-1,1),1)+diag(ones(nx-1,1),-1);
  Ay=-2*eye(ny)+diag(ones(ny-1,1),1)+diag(ones(ny-1,1),-1);
  bx=[10  zeros(1,nx-2)  20];
  by=[10; zeros(ny-2,1); 20];

  t=0; dt=0.4;
  ex=ones(1,nx); ey=ones(ny,1); eyy=ones(Ny+1,1);
  U=100*ones(ny,nx);
  Utot=[10*eyy [10*ex; U; 20*ex] 20*eyy];
```

```
qx=0.5*beta*dt/hx^2; Cx=eye(nx)-qx*Ax;
qy=0.5*beta*dt/hy^2; Cy=eye(ny)-qy*Ay;

while max(max(U))>37
% Bakåteuler i yled och Eulers metod i xled ett halvt tidssteg
  w=qy*by+qx*10*ey+(1-2*qx)*U(:,1)+qx*U(:,2);
  v=Cy\w; V=v;
  for j=2:nx-1
    w=qy*by+qx*U(:,j-1)+(1-2*qx)*U(:,j)+qx*U(:,j+1);
    v=Cy\w; V=[V v];
  end
  w=qy*by+qx*U(:,nx-1)+(1-2*qx)*U(:,nx)+qx*20*ey;
  v=Cy\w; V=[V v];
  Vtot=[10*eyy [10*ex; V; 20*ex] 20*eyy];
  t=t+dt/2;

% Bakåteuler i xled och Eulers metod i yled ett halvt tidssteg
  wT=qx*bx+qy*10*ex+(1-2*qy)*V(1,:)+qy*V(2,:);
  v=Cx\wT'; U=v';
  for i=2:ny-1
    wT=qx*bx+qy*V(i-1,:)+(1-2*qy)*V(i,:)+qy*V(i+1,:);
    v=Cx\wT'; U=[U; v'];
  end
  wT=qx*bx+qy*V(ny-1,:)+(1-2*qy)*V(ny,:)+qy*20*ex;
  v=Cx\wT'; U=[U; v'];
  Utot=[10*eyy [10*ex; U; 20*ex] 20*eyy];
  t=t+dt/2;

  subplot(1,2,1)
  imagesc(Utot), axis image, axis off, colorbar('vert')

  subplot(1,2,2)
  contour(X,Y,Utot,12)
  title(['t=' num2str(t)]), axis equal, drawnow
end
hold on
plot([0 Lx Lx 0 0],-[0 0 Ly Ly 0]), axis off
tslut=t, Utot
```

10 Numerisk behandling av elliptiska PDE

10.1 Poissons ekvation och Laplaces ekvation

En vanligt förekommande elliptisk differentialekvation är Poissons ekvation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = F(x, y) \tag{49}$$

som ofta skrivs $\nabla^2 u = F(x, y)$. (∇^2 kallas laplaceoperatorn.)
I elektrostatiska potentialproblem är $F(x, y)$ proportionell mot den givna laddningstätheten, i stationära värmeledningsproblem är $F(x, y)$ proportionell mot känd värmeutveckling i punkten (x, y).

Poissons ekvation och specialfallet när $F \equiv 0$, Laplaces ekvation, är de mest kända exemplen på elliptiska partiella differentialekvationer.

För att få en lösning till differentialekvationen ovan behövs randvillkor. Vanligtvis gäller att u eller $\partial u/\partial n$ (dirichletvillkor resp neumannvillkor) eller någon linjärkombination av u och $\partial u/\partial n$ (generaliserat neumannvillkor eller robinvillkor) är föreskrivna på randen till ett slutet område D.

Här betyder $\partial u/\partial n$ derivatan av u tagen i normalriktningen till randkurvan av området D. Lösningen önskas för punkter i det inre av D.

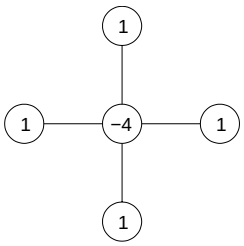
10.1.1 Fempunktsoperatorn och FDM

Vi använder finitadifferensmetoden (FDM) för den numeriska behandlingen. Betrakta ett rutnät med rumssteget h_x i x-led (horisontell led) och h_y i y-led (vertikal led). Den enklaste differensapproximationen till $\nabla^2 u$ är fempunktsoperatorn ∇_5^2,

$$\nabla_5^2 u_{ij} = \frac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{h_x^2} + \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h_y^2}.$$

För ett kvadratiskt rutnät med $h_x = h_y = h$ blir fempunktsoperatorn enklare:

$$h^2 \nabla_5^2 u_{ij} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}.$$



Figuren visar beräkningsmolekylen för $h^2 \nabla_5^2$.

Index i är radindex och j är kolumnindex. Låt oss anta att randen för området D består av räta linjer som antingen är parallella med koordinataxlarna eller bildar vinkeln 45^o med axlarna såsom i figuren. Ett kvadratiskt rutnät

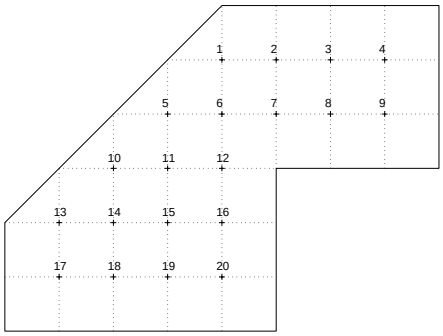
är lämpligt i detta fall, och om vi låter ∇^2 approximeras av ∇_5^2, så ersätts (49) av differensekvationen

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij} = h^2 F(x_j, y_i). \tag{50}$$

Eftersom differensekvationen baseras på differensapproximationer till andraderivatorna med trunkeringsfel $O(h^2)$, så kan man förvänta att trunkeringsfelet i lösningen är $O(h^2)$. Det beror emellertid även på den numeriska hanteringen av de givna randvillkoren.

Numrera punkterna i nätet enligt figuren och bilda en vektor $\mathbf{u}$ av de okända funktionsvärdena i gitterpunkterna. Heltalsparet $(i,\ j)$ ersätts av numret på gitterpunkten. Differensekvationen kan då skrivas som ett linjärt ekvationssystem

$$\mathbf{Au} = \mathbf{g}, \quad \mathbf{u} = (u_1, u_2, \ldots, u_N)^T.$$



För enkelhets skull ska vi begränsa diskussionen till så kallade dirichletproblem, då värdena på u är föreskrivna på randen. Då gäller följande:

1. Vektorn $\mathbf{g}$ består dels av kända värden $h^2 F_{ij} = h^2 F(x_j, y_i)$, dels av givna randvärden på u.

2. Ingen punkt har mer än fyra grannar. Alltså kommer matrisen $\mathbf{A}$ att ha högst fem element per rad. Alla diagonalelement a_{pp} i $\mathbf{A}$ är -4.

3. Om punkt p är granne till punkt q så gäller $a_{pq} = 1$, annars gäller $a_{pq} = 0$. Ur detta följer att $a_{pq} = a_{qp}$, dvs matrisen $\mathbf{A}$ är symmetrisk.

En typisk struktur visas här, $\mathbf{A}$ är som synes en bandmatris. I detta fall är $\mathbf{A}$ den 20×20-matris som erhålls för området med numreringen ovan. För två grannpunkter p och q gäller att $|p - q| \leq 6$, t ex har den sjätte punkten i området grannpunkter med nummer 1, 5, 7 och 12. Matrisens sjätte rad har därför ettor i kolumnerna 1, 5, 7 och 12. Samtliga diagonalelement är -4.

```
-4  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1 -4  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  1 -4  1  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  1 -4  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0 -4  1  0  0  0  0  1  0  0  0  0  0  0  0  0  0
 1  0  0  0  1 -4  1  0  0  0  0  1  0  0  0  0  0  0  0  0
 0  1  0  0  0  1 -4  1  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  1  0  0  0  1 -4  1  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  1  0  0  0  1 -4  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0 -4  1  0  0  1  0  0  0  0  0  0
 0  0  0  0  1  0  0  0  0  1 -4  1  0  0  1  0  0  0  0  0
 0  0  0  0  0  1  0  0  0  0  1 -4  0  0  0  1  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0 -4  1  0  0  1  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  1  0 -4  1  0  0  1  0  0
 0  0  0  0  0  0  0  0  0  0  1  0  0  1 -4  1  0  0  1  0
 0  0  0  0  0  0  0  0  0  0  0  1  0  0  1 -4  0  0  0  1
 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0 -4  1  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1 -4  1  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1 -4  1
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  1 -4
```

Om steget h halveras blir det generellt sett en ungefärlig fördubbling av bandbredden, medan antalet rader och kolumner i matrisen approximativt fyrfaldigas. I vårt exempel kommer antalet obekanta att öka från 20 till 101 vid halvering av h (verifiera detta), och $\mathbf{A}$ får därmed 101×101 element.

Gausselimination kan utnyttjas om ekvationssystemet inte är för stort. För att PDE-lösningen ska få god noggrannhet krävs oftast så litet steg h att ekvationssystemet får många hundra, kanske flera tusen, obekanta. Då är det nödvändigt ur både lagrings- och beräkningssynpunkt att tillämpa en effektivare lösningsmetod.

Eftersom matrisen är mycket gles är MATLABs `sparse`-finesser att rekommendera. Konjugeradegradientmetoden är också en effektiv algoritm för lösning av stora linjära ekvationssystem med gles och symmetrisk matris.

Matrisen $\mathbf{A}$ är symmetrisk om det är dirichletvillkor runt hela randen.

Studera Heath avsnitt 11.3.

10.1.2 Algoritm för FDM på Poissons ekvation

- Inneslut området i en rektangel med sidorna L_x och L_y och välj steglängd h. Antalet gitterpunkter i x-led respektive y-led uppgår då till $n_x = \frac{L_x}{h} + 1,\ n_y = \frac{L_y}{h} + 1$.
  I nätet i figuren på föregående sida gäller $n_x = 9$ och $n_y = 7$.

- Ta hand om randvärdena för u. Lägg in dem i en matris $\mathbf{S}$ med n_y rader och n_x kolumner med nollor överallt utom på platser (i, j) som representerar randen. Där ska gälla $S_{i,j} = u_{\text{rand}}$.
  (Om $u = 0$ runt hela randen kan denna fas hoppas över.)
  I figuren gäller att randvärden för u finns i 24 gitterpunkter, alltså på 24 platser i $\mathbf{S}$, till exempel i $S_{1,5}, S_{1,6}, S_{1,7}, S_{1,8}, S_{1,9}, S_{2,9}, S_{3,9}, S_{4,9}, S_{2,4}, S_{3,3}, S_{4,2}, S_{5,1}, S_{6,1}, S_{7,1}$.

- Skapa två vektorer $\mathbf{i}$ och $\mathbf{j}$ för numrering av de obekanta. Vektorn $\mathbf{i}$ innehåller radindex och $\mathbf{j}$ kolumnindex för de N numrerade punkterna.
  För vårt exempel kommer vektorerna att innehålla följande index:
  $\mathbf{i} = 2,2,2,2,\ 3,3,3,3,3,\ 4,4,4,\ 5,5,5,5,\ 6,6,6,6$
  $\mathbf{j} = 5,6,7,8,\ 4,5,6,7,8,\ 3,4,5,\ 2,3,4,5,\ 2,3,4,5.$

- Skapa också en behändig $n_y \times n_x$ matris $\mathbf{p}$ som uppfyller $p(i_k, j_k) = k$ för $k = 1, 2, ..., N$ och som är noll för övrigt.

- Nu kan matrisen och högerledsvektorn i ekvationssystemet $\mathbf{Au} = \mathbf{g}$ byggas upp med fempunktsformeln (50) och dess beräkningsmolekyl för $h^2\nabla_5^2$. Högerledsvektorn $\mathbf{g}$ innehåller dels h^2F_{ij}-värden, dels randvärden på u. Algoritmen för uppbyggnad av $\mathbf{A}$ och $\mathbf{g}$ kan i MATLAB formuleras så här:

```
A=-4*eye(N);
for k=1:N, i=I(k); j=J(k);
  g(k)=h^2*F(i,j);
  if p(i+1,j)>0, np=p(i+1,j); A(k,np)=1; else g(k)=g(k)-S(i+1,j); end
  if p(i-1,j)>0, np=p(i-1,j); A(k,np)=1; else g(k)=g(k)-S(i-1,j); end
  if p(i,j+1)>0, np=p(i,j+1); A(k,np)=1; else g(k)=g(k)-S(i,j+1); end
  if p(i,j-1)>0, np=p(i,j-1); A(k,np)=1; else g(k)=g(k)-S(i,j-1); end
end
```

  Man kan effektivisera algoritmen genom att utnyttja två vektorer `Arow` och `Acol` för att samla radindex och kolumnindex för elementen i $\mathbf{A}$ som utgörs av ettor — högst fyra per rad medför att högst $4N$ komponenter ingår i vardera vektorn `Arow` och `Acol`. Därefter kan matrisen skapas i en enda sats.
  För beräkningar med Laplaces ekvation, alltså då $F(x, y) = 0$, erhålls $\mathbf{A}$ och $\mathbf{g}$ av följande funktion:

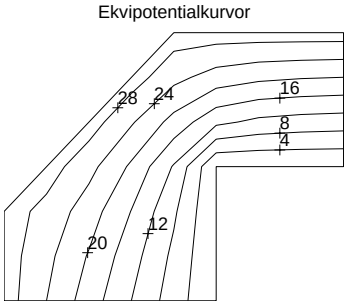
```
function [A,g]=fbyggAlp(I,J,S,ny,nx)
 % Bygger upp A och g i laplacefallet
 % Randvärdesmatrisen S har storleken (ny,nx)
  N=length(I);
  p=sparse(I,J,1:N,ny,nx);
  g=zeros(N,1);
  Arow=zeros(1, 4*N); Acol=Arow;   % reservera plats för 4*N ettor
  m=0;
  for k=1:N
    i=I(k); j=J(k);
    pN=p(i-1,j);  pS=p(i+1,j);  pW=p(i,j-1);  pE=p(i,j+1);
    if pN>0, m=m+1; Arow(m)=k; Acol(m)=pN; else g(k)=g(k)-S(i-1,j); end
    if pS>0, m=m+1; Arow(m)=k; Acol(m)=pS; else g(k)=g(k)-S(i+1,j); end
    if pW>0, m=m+1; Arow(m)=k; Acol(m)=pW; else g(k)=g(k)-S(i,j-1); end
    if pE>0, m=m+1; Arow(m)=k; Acol(m)=pE; else g(k)=g(k)-S(i,j+1); end
  end
  Arow=Arow(1:m); Acol=Acol(1:m);     % bort med de sista nollelementen
  A=-4*speye(N)+sparse(Arow,Acol,1);
```

- Lös ekvationssystemet $\mathbf{Au} = \mathbf{g}$ (effektivt med MATLABs `sparse`-finesser) och som resultat erhålls $\mathbf{u}$-vektorns N komponenter.
  Placera in de beräknade värdena på sin rätta plats i gittret. Rita slutligen resultatet på lämpligt sätt.

10.2 Laplaces ekvation för skivan

Bestäm med FDM en approximativ lösning till Laplaces ekvation $\nabla^2 u = 0$ för samma skiva som vi har betraktat tidigare. Skivan ryms i en rektangel med sidorna 8 och 6 längdenheter. Rutstorleken i gittret är $h = 1$.

På randen gäller följande dirichletvillkor: Vid vänstra randen, vid 45^o-randen och vid skivans övre rand är potentialen konstant, $u = 30$.
I det nedre högra partiet är potentialen noll. Potentialen antas sjunka linjärt från trettio till noll på de två övriga ränderna (den högra och den undre), se matrisen `S` i programmet.
Rita resultatet i form av ekvipotentialkurvor.




```
  h=1;  x=0:h:8; y=-(0:h:6);
  [X,Y]=meshgrid(x,y);
% Skapa randvärdesmatris S och indexvektorer I och J
  S=zeros(7,9);
  S(1,6:8)=30;
  for i=1:5, S(i,1:6-i)=30; end
  S(1:3,9)=[30 20 10]'; S(6,1)=30;
  S(7,1:5)=[30 24 18 12 6];
  I=[2 2 2 2 3 3 3 3 3 4 4 4 5 5 5 5 6 6 6 6];
  J=[5 6 7 8 4 5 6 7 8 3 4 5 2 3 4 5 2 3 4 5];
  N=length(I)
  [A,g]=fbyggAlp(I,J,S,7,9);
  u=A\g;

  U=S; for k=1:N, U(I(k),J(k))=u(k); end
  C=contour(X,Y,U,0:4:28), hold on
  title('Ekvipotentialkurvor'), axis off
  plot([4 8 8 5 5 0 0 4],-[0 0 3 3 6 6 4 0])   % rita skivans kontur
  clabel(C)
```

10.3 Utböjning av skivan

Beräkna utböjningen hos en tunn homogen skiva upplagd på en horisontell ram med samma kontur som tidigare. Skivan har elasticitetsmodulen E och tröghetsmomentet $I(x, y)$. Skivan tyngs ner av en tryckfördelning $q(x, y)$. Utböjningen $u(x, y)$ bestäms av differentialekvationerna

$$\nabla^2 M = -q(x, y), \quad \nabla^2 u = M(x, y)/EI(x, y)$$

båda av typen Poissons ekvation (49). Inga moment verkar vid skivans kanter och utböjningen vid ramen är noll. Det innebär att randvillkoren utgörs av

$M = 0$ och $u = 0$.

De båda differentialekvationerna ger upphov till samma matris, eftersom det dels är samma differentialoperator ∇^2, dels samma dirichletrandvillkor för båda. Eftersom randvärdena är noll blir randvärdesmatrisen en nollmatris. Ekvationssystemets högerled kommer enbart att bero av värdena i gitterpunkterna för differentialekvationens högerled.

Vi får därmed följande två ekvationssystem att lösa, det första för momentet $\mathbf{m}$ i skivans gitterpunkter och det andra för utböjningen $\mathbf{u}$ i samma punkter: $\mathbf{Am} = -h^2\,\mathbf{q}$ och $\mathbf{Au} = h^2\,\mathbf{m}/EI$.

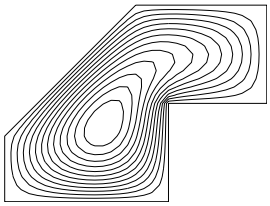
I vårt exempel gäller konstant tryckfördelning, $q = 20000$ och $EI = 10^5$. (Eftersom högerledet i Poissons ekvation har snällt utseende kan funktionen `[A,g]=fbyggAlp(...)` utnyttjas utan modifiering, `g`-vektorn blir noll. Motivera detta!)

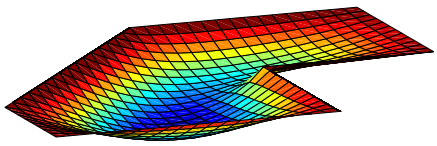
Förutom att räkna med steget $h = 1$ med 20 obekanta vill vi göra om FDM-beräkningarna med finare steg för att få en bättre approximation till differentialekvationsproblemets lösning. Med steget $h = 0.5$ blir det 101 obekanta och med steget $h = 0.25$ fyrfaldigas antalet obekanta ungefärligen och blir i detta fall $N = 449$.

Hur noggrannheten förbättras kan vi studera genom att notera maximal nedböjning i de tre beräkningsfallen: $u_{min} = -0.3410,\ -0.3449,\ -0.3488$. Med ytterligare en steghalvering blir det 1889 obekanta och nedböjningen $u_{min} = -0.3502$. Resultatet bedöms ha två säkra decimaler och kan anges $u_{min} = -0.350 \pm 0.002$.

```
% skiv1bojning
% Poissons ekvation två ggr,  M=0 och u=0 på randen
  clear, figure(1), clf
  xrand=[4 8 8 5 5 0 0 4];
  yrand=[0 0 3 3 6 6 4 0];
  h=1;
  for fall=1:3
    x=(0:h:8); nx=length(x);
    y=(0:h:6); ny=length(y);
    [X,Y]=meshgrid(x,-y);

  % Vänstra och högra randen
    yV1=(0:h:4)';   xV1=4-yV1;
    yV2=(4+h:h:6)'; xV2=0*yV2;
    xV=[xV1; xV2];

    yH1=(0:h:3-h)'; xH1=8*ones(size(yH1));
    yH2=(3:h:6)';   xH2=5*ones(size(yH2));
    xH=[xH1; xH2];
```

```
% Snygga till X-matrisen så att inga värden hamnar utanför skivan
  for i=1:length(yV1)-1
    j=find(X(i,:)<xV1(i)); X(i,j)=xV1(i);  % ta bort vänstra hörnet
  end
  for i=(3/h+2):ny, X(i,5/h+2:nx)=5; end   % ta bort högra kvadraten

% Numrera punkterna
  JV=round(xV/h)+1;
  JH=round(xH/h)+1;
  I=[]; J=[];
  for i=2:ny-1
    jrad=JV(i)+1 : JH(i)-1;
    irad=i*ones(size(jrad));
    I=[I irad]; J=[J jrad];
  end
  N=length(I)

  S=zeros(ny,nx);                % randvärdesmatrisen
  [A,g]=fbyggAlp(I,J,S,ny,nx);   % (kan användas i detta specialfall!)

  q=20000*ones(N,1);  EI=1e5;
  M=A\(-h^2*q);
  u=A\(h^2*M/EI);
  umin=min(u)                    % största utböjning

  U=S;
  for k=1:N, U(I(k),J(k))=u(k); end
  contour(X,Y,U,12), hold on, axis equal
  plot(xrand,-yrand), axis off
  pause(1)
  hold off

  h=h/2;
end
figure(2), clf, subplot(2,1,1)
surf(X,Y,U), axis([0 8 -6 1 -0.5 0]), axis off
```





10.4 Numerisk behandling av egenvärdesproblem för PDE

Ett närbesläktat problem till randvärdesproblem för elliptiska PDE såsom Poissons ekvation är egensvängningsproblemet för differentialekvationen

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \omega^2 u = 0 \tag{51}$$

med villkoret $u = 0$ eller $\partial u/\partial n = 0$ på randen till det slutna området D. Vi begränsar oss här till randvillkoret $u = 0$. En trivial lösning till problemet är $u(x, y) = 0$ i hela området D, men för vissa värden på ω, de så kallade egenfrekvenserna, finns icketriviala lösningar.

Med finitadifferensmetoden och ∇^2 approximerad av fempunktsoperatorn ∇_5^2 kan (51) omformas till

$$\frac{1}{h^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}) + \omega^2 u_{ij} = 0. \tag{52}$$

På samma sätt som tidigare numrerar vi punkterna i nätet och ersätter heltalsparet (i, j) av numret på gitterpunkten. Differensekvationen (52) kan nu skrivas i matrisform som $\frac{1}{h^2}\mathbf{Au} = -\omega^2\mathbf{u}$.

Om vi inför $\lambda = -\omega^2$ erhålls matrisegenvärdesproblemet $\frac{1}{h^2}\mathbf{Au} = \lambda\mathbf{u}$. Matrisen $\mathbf{A}$ är samma glesa matris som tidigare — konstruerad enligt algoritmen i avsnitt 10.1.2. Med lämplig algoritm beräknas egenvärden och egenvektorer, och vi får på så sätt en approximativ lösning till egenvärdesproblemet för differentialekvationen. Med en mycket tät diskretisering, det vill säga ett finmaskigt rutnät med många gitterpunkter, bör egenfrekvenserna kunna bestämmas med god precision.

När det gäller vibrationsproblem för t ex membran eller tunna homogena skivor, är det intressantast att studera de tre eller fyra lägsta egenfrekvenserna och deras egensvängningsmoder. Vad kan det motsvara i det approximerande matrisegenvärdesproblemet? Är det matrisens största eller minsta egenvärden som ska beräknas?

Sambandet mellan ω och λ lyder $\omega = \sqrt{-\lambda}$. Att ha $-\lambda$ under rottecknet är meningsfullt enbart om egenvärdet är reellt och negativt. Med hjälp av symmetriegenskapen och gerschgorincirklar inser man att alla egenvärden till $\frac{1}{h^2}\mathbf{A}$ är reella och ickepositiva (verifiera detta!). Det är det till beloppet minsta av egenvärdena till $\mathbf{C}$ som är av intresse och som vi alltså ska beräkna. Egenfrekvenserna bestäms därefter av $\omega_1 = \sqrt{-\lambda_1},\ \omega_2 = \sqrt{-\lambda_2}, \ldots$

Algoritmen bör också beräkna egenvektorn till egenvärdet λ_k, eftersom det är egenvektorn som bestämmer formen hos egensvängningsmoden som tillhör egenfrekvensen ω_k i differentialekvationen.

10.4.1 Effektiv beräkning av egenvärden och egenvektorer

Rent praktiskt är egenvärdes- och egenvektorsbestämningen för en PDE ett krux eftersom det oftast är fråga om stora matriser. Att använda `eig(A)` går bra till exempel för fallet $N = 20$.

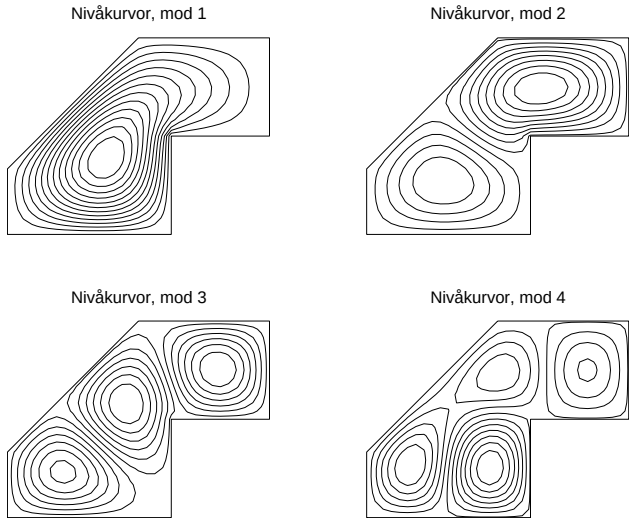
För större `sparse`-lagrade matriser kan `[V,Lambda]=eigs(A,k,'sm')` tillämpas som beräknar de k minsta egenvärdena och tillhörande egenvektorer (k kolumner i V-matrisen). Parametern 'sm' betyder small; gör `help eigs` för mer information.

Vid mycket stora glesa matriser är `eigs`-kommandot ovan tidskrävande. En uppsnabbande algoritm är att enbart bestämma egenvärdena med eigs och därefter utnyttja metoden som beskrivs i nästa avsnitt för att beräkna tillhörande egenvektorer. Ett gott alternativ för beräkning av ett enstaka egenvärde är inversa potensmetoden med skift. Då erhålls samtidigt egenvektorn.

```
% skiv1svangning
% Egensvängningsmoder för skivan
% Se programmet skiv1bojning t o m satsen [A,g]=...
% Tre fall (steghalveringar) studeras:  h = 0.5, 0.25, 0.125
  ... ... ...

  % Bestäm de fyra minsta egenvärdena till A/h^2  med eigs
    Nlamb=4;
    opts.disp=0;        % undertrycker utskrift av iterationer
    [V,Lamb]=eigs(A/h^2,Nlamb,'sm',opts);
    lamb=diag(Lamb)';
    omega=sqrt(-lamb)
    h=h/2;
  end

% Rita nivåkurvor vid finaste gittret
  for mod=1:Nlamb
    v=V(:,mod);
    u=v/norm(v,inf);                  % gör maxkomp=1
    U=S;
    for k=1:N, U(I(k),J(k))=u(k); end
    subplot(2,2,mod)
    contour(X,Y,U,12),  hold on
    plot(xrand,-yrand,'k'), axis equal
    title(['Nivåkurvor, mod ' int2str(mod)]), axis off
  end
```

Här är de fyra lägsta egenfrekvenserna vid allt finare gitter. Figurerna visar de fyra egensvängningsmodernas nivåkurvor.

```
h=0.5    N= 101,   w = 0.9758  1.1861  1.4276  1.5966
h=0.25   N= 449,   w = 0.9773  1.1934  1.4437  1.6165
h=0.125  N=1889,   w = 0.9771  1.1946  1.4475  1.6214
```



10.4.2 Egenvektor till känt egenvärde

För ett egenvärde och dess tillhörande egenvektor finns sambandet

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \implies (\mathbf{A}-\lambda\mathbf{I})\mathbf{x} = \mathbf{0}.$$

Låt $\mathbf{C}$ beteckna matrisen $\mathbf{A}-\lambda\mathbf{I}$. Om $\mathbf{x}$ är en lösning, så är också $\alpha\mathbf{x}$ för godtyckligt α en lösning till det homogena systemet $\mathbf{C}\mathbf{x} = \mathbf{0}$. Vi väljer en lösning sådan att $x_n = -1$; egenvektorn är alltså $\mathbf{x}^T = (x_1, x_2, ..., x_{n-1}, -1)$. Vi inför $\mathbf{x}^{(n-1)}$ för de $n-1$ första komponenterna. Partitionera matrisen så att systemet kan skrivas

$$\begin{pmatrix} \mathbf{C}_{n-1} & \mathbf{c}_n^{(n-1)} \\ (c_{n1}, c_{n2}, ...) & c_{nn} \end{pmatrix} \begin{pmatrix} \mathbf{x}^{(n-1)} \\ -1 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix} \implies \mathbf{C}_{n-1}\mathbf{x}^{(n-1)} + \mathbf{c}_n^{(n-1)}(-1) = \mathbf{0}$$

Genom lösning av systemet $\mathbf{C}_{n-1}\mathbf{x}^{(n-1)} = \mathbf{c}_n^{(n-1)}$ erhålls egenvektorns $n-1$ första komponenter. Sista komponenten är -1, därmed är egenvektorn känd. Den kan sedan normeras på vanligt sätt så att $\|\mathbf{x}\|_2 = 1$ eller $\|\mathbf{x}\|_\infty = 1$.

Alternativ algoritm för egensvängningsproblemet; låt oss i exemplet ta fallet $h = 0.125$. Med `eigs` på sparsematrisen fås egenvärdena (här de sex minsta), och sedan erhålls egenvektorn genom sparselösning på ekvationssystemet i egenvektorsalgoritmen.

```
  h=0.125;
% Skapa gitter och bygg upp sparsematrisen A som förut
   ... ... ...
% Bestäm de sex minsta egenvärdena till A/h^2 med eigs
  Nlamb=6;
  opts.disp=0;
  lamb=eigs(A/h^2,Nlamb,'sm',opts);
  omega=sqrt(-lamb)

% Beräkna tillhörande egenvektorer
  for mod=1:Nlamb
    C=A/h^2-lamb(mod)*speye(N);
    Cp=C(1:N-1,1:N-1);
    b=C(1:N-1,N);
    v=Cp\b;  v=[v;-1];        % egenvektorn v
    u=v/max(abs(v));
    U=S;
    for k=1:N, U(I(k),J(k))=u(k); end
  % Rita nivåkurvor som förut
    ... ... ...
  end
```

10.5 Ickelinjära elliptiska PDE

Låt oss studera Poissons ekvation (49) men nu med ett högerled $F(u)$ som är en ickelinjär funktion av u,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = F(u) \quad \text{med } F(u) = 4\sqrt{u}.$$

Vi önskar bestämma potentialfördelningen $u(x, y)$ för samma skiva och med samma randvillkor som i Laplace ekvation i avsnitt 10.2. Det som skiljer vårt nya problem från laplaceproblemet är tillkomsten av den ickelinjära termen i högerledet.

Vid studium av algoritmen i avsnitt 10.1.2 (speciellt de sista punkterna) inses att ekvationssystemet nu kommer att lyda $\mathbf{Au} = \mathbf{g} + h^2\mathbf{F}$, där $\mathbf{A}$ och $\mathbf{g}$ är samma som i laplaceproblemet och där vektorn $\mathbf{F}$ består av komponenterna $F(u_i) = 4\sqrt{u_i},\ i = 1, 2, ..., N$ (u_i är de omnumrerade obekanta).

Newtons metod kan tillämpas på detta ickelinjära system om vi skriver det på formen $\mathbf{f}(\mathbf{u}) = \mathbf{Au} - \mathbf{g} - h^2\,\mathbf{F} = 0$, med jacobianen $\mathbf{J} = \mathbf{A} - h^2\,\mathbf{F}'$, där $\mathbf{F}'$ är en diagonalmatris med diagonalelementen $F'(u_i) = 2/\sqrt{u_i}$.

Jacobianen får precis samma glesa struktur som $\mathbf{A}$, därför bör $\mathbf{J}$ skrivas som sparsematris om antalet obekanta är stort. I varje iteration i Newtons

metod löses det linjära systemet $\mathbf{J}\,\delta\mathbf{u} = -\mathbf{f}$, varefter $\mathbf{u}$-vektorn uppdateras med $\delta\mathbf{u}$. God startgissning till den okända vektorn $\mathbf{u}$ krävs, vilket ofta är ett krux.

I algoritmen nedan löser vi först det linjära laplaceproblemet (då $F(u) = 0$). Denna lösning utnyttjas som startgissning för det ickelinjära poissonproblemet. Det blir snabb konvergens, bara sex iterationer krävs.

Om olinjariteten är kraftigare än här, kan man behöva utnyttja någon form av inbäddning för att hitta en startgissning som ger konvergens.

Det innebär att man placerar en faktor κ framför den olinjära termen och låter först κ var litet (strax över noll); beräknar en lösning; utnyttjar denna som startgissning då κ gjorts lite större. Fortsätt så tills κ når värdet ett, förhoppningsvis har man då en konvergerande lösning till det ickelinjära ursprungsproblemet.

```
% Bygg upp A, g och S enligt laplace-exemplet (h=1)
  ... ... ...
  u=A\g; dunorm=1;  iter=0;
  while dunorm>1e-8 & iter<10
    F=4*sqrt(u); Fprim=2./sqrt(u);
    f=A*u-g-h^2*F;  Jacob=A-h^2*diag(Fprim);
    du=-Jacob\f; dunorm=norm(du), u=u+du; iter=iter+1;
  end
  U=S; for k=1:N, U(I(k),J(k))=u(k); end
  % Rita med ekvipotentialkurvor med contour, se laplacekoden
```

10.6 Parabolisk PDE i två rumsvariabler igen

I avsnitt 9.7 tog vi upp några metoder för numerisk behandling av värmeledningsproblem i 2D-fallet. Temperaturvariationen $u(x, y, t)$ beskrivs av värmeledningsekvationen (46), alltså

$$\frac{\partial u}{\partial t} = \beta \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

där β är värmediffusiviteten i området som har ett känt tvärsnitt i xy-planet. Randvillkoren är att temperaturen runt hela området är bekant (dirichletvillkor) eller att det finns ett känt derivatasamband på någon del av randen (neumannvillkor). Begynnelsevillkoret är att temperaturen i hela området är känd vid tiden $t = 0$.

En algoritm som lämpar sig utmärkt för områden med rektangulärt tvärsnitt är alterneranderiktningsmetoden (ADI-metoden) som behandlades i avsnitt 9.7. Om tvärsnittet har polygonform av den typ som skivan i avsnitt 10.2, kan ADI-metoden inte tillämpas.

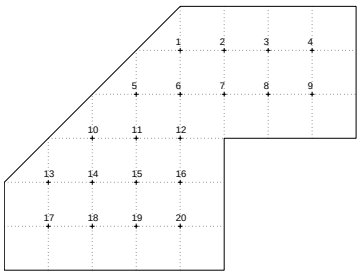
För paraboliska problem i en rumsdimension är Crank-Nicolsons metod den populäraste algoritmen. Den blir inte så besvärlig att utvidga till 2D-fallet, åtminstone inte om vi har ett kvadratiskt rutnät så att fempunktsoperatorn ∇_5^2 får sin enklaste form. Differentialekvationen (46) övergår till ett system av ordinära differentialekvationer enligt formel (47), som för fallet $h_x = h_y = h$ förenklas till

$$\frac{du_{ij}}{dt} = \frac{\beta}{h^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij})$$

för alla punkter i rutnätet. Vi numrerar punkterna i nätet precis som i algoritmen för FDM på Poissons ekvation. Det innebär att systemet kan skrivas på matrisform

$$\frac{d\mathbf{u}}{dt} = \gamma\,(\mathbf{A}\mathbf{u} + \mathbf{b}(t)), \quad \mathbf{u}(0) = \mathbf{u}_{start}, \quad \text{med } \gamma = \beta/h^2.$$

Låt oss tillämpa Crank-Nicolsons metod (32) för att studera nedkylningen av ett 100 grader varmt block med värmediffusiviteten $\beta = 0.1$. För blocket gäller tvärsnitt och diskretisering enligt figuren. Randtemperaturen antas för enkelhets skull vara oberoende av tiden. Låt samma randvärden gälla som vi hade i Laplaces ekvation för skivan (avsnitt 10.2).



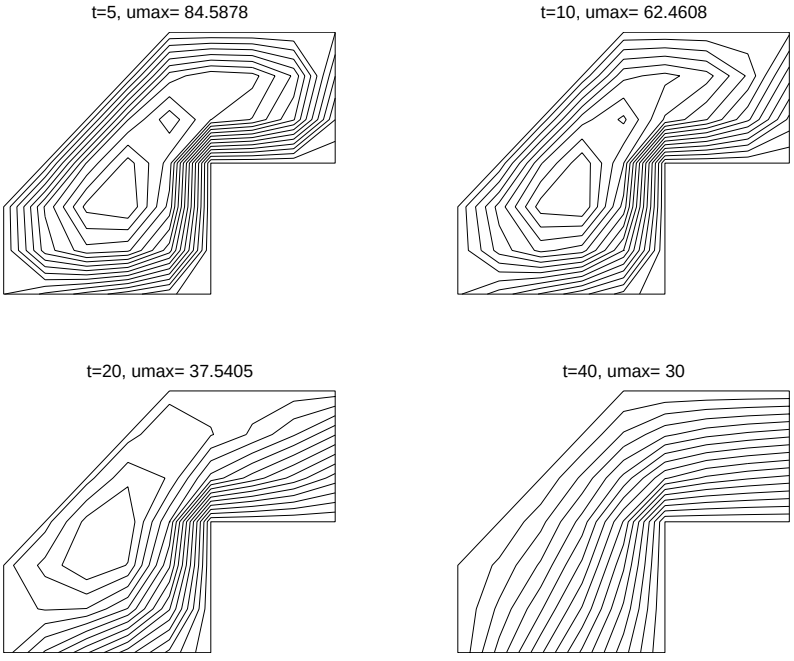
Vid vänstra randen, vid 45^o-randen och vid övre randen är temperaturen konstant 30^oC. I det nedre högra partiet är temperaturen noll. Längs de två övriga ränderna avtar temperaturen linjärt från trettio till noll grader.

Matrisen är samma som i laplaceproblemet för skivan. Vektorn $\mathbf{b}(t)$ är uppbyggd av de i vårt fall tidsoberoende randvärdena och uppfyller $\mathbf{b} = -\mathbf{g}$, där $\mathbf{g}$ utgörs av högerledsvektorn i laplaceproblemet.

En MATLAB-algoritm för Crank-Nicolsons metod har vi träffat på i programmet `varmcranknic` tidigare, och den kommer till användning nu igen.

Programmet `varmeskiva` börjar med att vara identiskt med koden för laplaceproblemet i avsnitt 10.2 fram till dess att matrisen $\mathbf{A}$ och vektorn $\mathbf{g}$ har skapats. Värmeledningsproblemet löser vi sedan med tidssteget $\delta t = 1$ i 40 tidssteg. I varje tidssteg ritas ögonblicksbilder (isotermer) över temperaturvariationen i området.

```
% varmeskiva, du/dt = beta (d2u/dx2 + d2u/dy2),  polygontvärsnitt
  h=1;  x=0:h:8; y=-(0:h:6); [X,Y]=meshgrid(x,y);
  xrand=[4 8 8 5 5 0 0 4]; yrand=-[0 0 3 3 6 6 4 0];
% Bygg upp A, g och S enligt laplace-exemplet
    ...  ...  ...
% Crank-Nicolsons metod för lösning av du/dt=gamma(Au+b)
  beta=0.1; gamma=beta/h^2;  b=-g;
  t=0; u=100*ones(N,1);                    % begynnelsetemperatur 100 grader
  U=S; for k=1:N, U(I(k),J(k))=u(k); end
  subplot(2,2,4), contour(X,Y,U,16), hold on, plot(xrand,yrand,'k')
  axis off, title(['t=' num2str(t)])

  dt=1; q=dt*gamma/2; C=eye(N)-q*A;
  while t<40
    w=u+q*(A*u+2*b); u=C\w; t=t+dt;
    U=S; for k=1:N, U(I(k),J(k))=u(k); end
    subplot(2,2,4), hold off, contour(X,Y,U,16), hold on
    plot(xrand,yrand,'k'), axis off, title(['t=' num2str(t)]), drawnow
    if t==5 | t==10 | t==20
      subplot(2,2,1+fix(t/10))
      contour(X,Y,U,16), hold on, plot(xrand,yrand,'k')
      axis off, title(['t=' int2str(t) ', umax= ' num2str(max(max(U)))])
    end
  end
  title(['t=' int2str(t) ', umax= ' num2str(max(max(U)))])
```



10.7 Laplaceoperatorn i andra koordinatsystem

När man vill studera potentialfördelningen eller lösa värmeledningsproblem i en cylinder eller annan kropp som är rotationssymmetrisk kring z-axeln är det lämpligt att utnyttja cylindriska koordinater $(r,\, z)$.

Om området utgörs av en cirkulär eller kanske njurformad skiva med vinkelberoende egenskaper är en framställning i polära koordinater $(r,\, \varphi)$ bäst. Och om vi önskar lösa en PDE i en sfär eller halvsfär är det naturligtvis lämpligast med sfäriska koordinater.

Vi behöver därför känna till hur laplaceoperatorn ser ut i olika koordinatsystem och kunna ställa upp en approximerande fempunktsformel med beräkningsmolekyl analogt med formel (50).

10.7.1 Cylindriska koordinater

Låt oss betrakta Poissons ekvation $\nabla^2 u = F(r,z)$ i cylinderkoordinater (utan φ-beroende):

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\,\frac{\partial u}{\partial r} + \frac{\partial^2 u}{\partial z^2} = F(r,z). \tag{53}$$

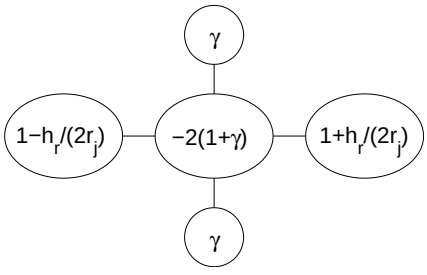
Vi lägger ett gitter med steget h_r i r-led och steget h_z i z-led. Med de vanliga centraldifferenskvoterna för derivatorna approximeras differentialekvationen av följande uttryck:

$$\frac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{h_r^2} + \frac{u_{i,j+1} - u_{i,j-1}}{2r_j h_r} + \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h_z^2} = F(r_j, z_i).$$

Multiplicera med h_r^2 på båda sidor och inför beteckningen $\gamma = h_r^2/h_z^2$.
Då kan differensapproximationen till Poissons ekvation skrivas

$$(1-\frac{h_r}{2r_j})u_{i,j-1}+(1+\frac{h_r}{2r_j})u_{i,j+1}+\gamma(u_{i-1,j}+u_{i+1,j})-2(1+\gamma)u_{i,j} = h_r^2 F(r_j,z_i) \tag{54}$$

Beräkningsmolekylen för vänsterledet ser ut så här:



Att r finns i nämnaren i laplaceoperatorn och dess fempunktsapproximation ställer till lite problem om det betraktade området innefattar $r = 0$.

För en rotationssymmetrisk kropp med rörform gäller att r är större än noll överallt och då är formel (54) tillämpbar i alla gitterpunkter. Men om cylinderkroppen är kompakt utan hål runt z-axeln måste vi undersöka vad som händer vid $r = 0$. Av (rotations-)symmetriskäl gäller att $\partial u/\partial r = 0$ där. Den andra termen i laplaceoperatorn i formel (53) blir då $0/0$, vilket innebär att vi får användning av l'Hospitals regel. Vad sker då? Jo, just vid $r = 0$ kommer differentialekvationen att övergå till

$$2\frac{\partial^2 u}{\partial r^2} + \frac{\partial^2 u}{\partial z^2} = F(0, z). \tag{55}$$

(Härled detta!)
I diskretiseringen i r-led finns index $j = 1$ vid randen $r = 0$. Formel (55) och symmetrin kring z-axeln leder till följande differensapproximation:

$$4u_{i,2} + \gamma(u_{i-1,1} + u_{i+1,1}) - 2(2 + \gamma)u_{i,1} = h_r^2 F(0, z_i).$$

Därmed har vi kommit en god bit på vägen i finitadifferensmetoden för lösning av Poissons ekvation i cylinderkoordinater.

För övrigt gäller det att följa algoritmen i avsnitt 10.1.2 för FDM på Poissons ekvation fast nu i en modifierad version med hänsyn tagen till cylinderfallets fempunktsformel och tillhörande beräkningsmolekyl.

10.7.2 Polära koordinater

I plana polära koordinater lyder Poissons ekvation

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial \varphi^2} = F(r, \varphi).$$

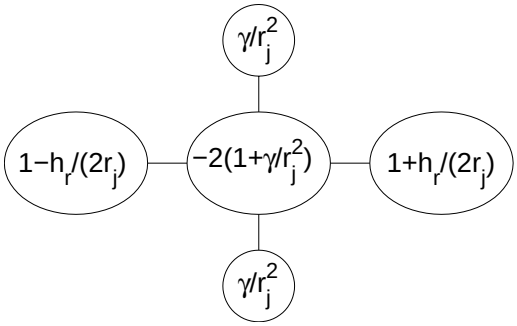
Som synes påminner denna ekvation mycket om formel (53). Därför kommer även differensapproximationen att vara snarlik (54). I radiell led är diskretiseringssteget liksom tidigare h_r.

I φ-led låter vi $\delta\varphi$ utgöra en liten vinkelförändring, (t.ex. $\delta\varphi = 2\pi/30$).

Inför beteckningen $\gamma = h_r^2/(\delta\varphi)^2$. Då kan differensapproximationen i polära fallet skrivas

$$(1-\frac{h_r}{2r_j})u_{i,j-1}+(1+\frac{h_r}{2r_j})u_{i,j+1}+\frac{\gamma}{r_j^2}(u_{i-1,j}+u_{i+1,j})-2(1+\frac{\gamma}{r_j^2})u_{i,j} = h_r^2 F(r_j, \varphi_i) \tag{56}$$

Beräkningsmolekylen för vänsterledet får följande utseende:



Precis som i cylinderfallet blir det lite problematiskt om $r = 0$ råkar vara en inre punkt i det betraktade området. Detta åtgärdas här på samma sätt som i föregående avsnitt.

När man vill lösa Laplaces ekvation i ett område där polära koordinater passar bäst kan följande funktion vara användbar. Den är en motsvarighet till funktionen `fbyggAlp` för cartesiska koordinater i avsnitt 10.1.2.

I polära fallet är diagonalelementen inte -4 utan beroende av r. Därför reserveras nu utrymme i vektorerna även för dessa; det innebär totalt $5N$ ickenollelement enligt fempunktsformeln.

```
function [A,g]=fbyggpolarlp(I,J,S,nfi,nr,dfi,hr,r)
 % Polära koordinater, bygger upp A och g i laplacefallet
 % Randvärdesmatrisen S har storleken (nfi,nr)
  N=length(I); p=sparse(I,J,1:N,nfi,nr);
  g=zeros(N,1);
  Arow=zeros(1, 5*N);   % reservera plats för 5N element (diagonal)
  Acol=Arow; Aval=Arow;
  m=0;
  for k=1:N,  i=I(k); j=J(k);
    gam=(hr/dfi)^2;
    cg=gam/r(j)^2;
    cdr=hr/(2*r(j));
    m=m+1; Arow(m)=k; Acol(m)=k; Aval(m)=-2*(1+cg);  % diagonalelement
    if p(i+1,j)>0, m=m+1; Arow(m)=k; Acol(m)=p(i+1,j); Aval(m)=cg;
      else g(k)=g(k)-S(i+1,j)*cg; end
    if p(i-1,j)>0, m=m+1; Arow(m)=k; Acol(m)=p(i-1,j); Aval(m)=cg;
      else g(k)=g(k)-S(i-1,j)*cg; end
    if p(i,j+1)>0, m=m+1; Arow(m)=k; Acol(m)=p(i,j+1); Aval(m)=1+cdr;
      else g(k)=g(k)-S(i,j+1)*(1+cdr); end
    if p(i,j-1)>0, m=m+1; Arow(m)=k; Acol(m)=p(i,j-1); Aval(m)=1-cdr;
      else g(k)=g(k)-S(i,j-1)*(1-cdr); end
  end
  A=sparse(Arow(1:m),Acol(1:m),Aval(1:m),N,N);
```

10.7.3 Sfäriska koordinater

Låt oss nu betrakta Poissons ekvation i sfäriska koordinater (r, θ, φ), men liksom i fallet med cylindriska koordinater inskränker vi oss till φ-oberoende tillämpningar och skippar alltså den tredje variabeln φ.

Vänsterledet (laplaceoperatorn $\nabla^2 u$) kan i sfäriska fallet med koordinaterna r och θ uttryckas på olika sätt:

$$\nabla^2 u = \frac{1}{r}\frac{\partial^2 (ur)}{\partial r^2} + \frac{1}{r^2 \sin\theta}\frac{\partial}{\partial \theta}\left(\sin\theta \frac{\partial u}{\partial \theta}\right)$$

eller

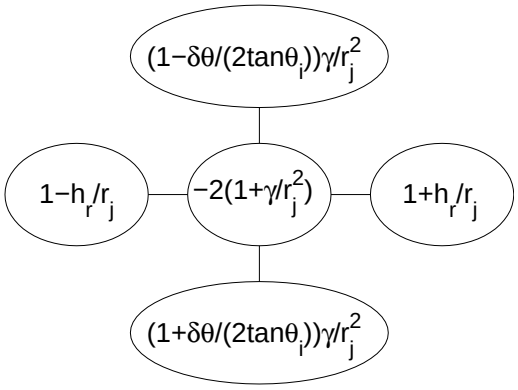
$$\nabla^2 u = \frac{\partial^2 u}{\partial r^2} + \frac{2}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\left(\frac{\partial^2 u}{\partial \theta^2} + \frac{1}{\tan\theta}\frac{\partial u}{\partial \theta}\right)$$

Derivering av produkterna i det övre uttrycket leder till det undre uttrycket. Visa detta!

Vinkeln θ är noll vid nordpolen och $\theta = \pi$ vid sydpolen.

Finitadifferensmetoden på Poissons ekvation börjar med diskretisering i r-led med steget h_r och i θ-led med $\delta\theta$. Med differenskvoter för derivatorna som tidigare och med beteckningen $\gamma = h_r^2/(\delta\theta)^2$ erhålls följande differensapproximation till Poissons ekvation:

$$(1-\frac{h_r}{r_j})u_{i,j-1}+(1+\frac{h_r}{r_j})u_{i,j+1}+\frac{\gamma}{r_j^2}(1-\frac{\delta\theta}{2\tan\theta_i})u_{i-1,j}+\frac{\gamma}{r_j^2}(1+\frac{\delta\theta}{2\tan\theta_i})u_{i+1,j}-$$
$$-2(1+\frac{\gamma}{r_j^2})u_{i,j} = h_r^2 F(r_j, \theta_i) \qquad (57)$$



Förutom att det kan trassla vid $r = 0$, måste åtgärder också vidtas vid $\theta = 0$ och $\theta = \pi$, eftersom nämnarna som innehåller $\tan\theta$ blir noll vid dessa vinklar. Där gäller av symmetriskäl det naturliga randvillkoret $\partial u/\partial\theta = 0$, och l'Hospitals regel hjälper oss att få bukt med problemet.

11 Numerisk behandling av hyperboliska PDE

11.1 Vågekvationen

Den vanligaste hyperboliska partiella differentialekvationen är vågekvationen:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}, \quad 0 \le x \le L, \;\; t \ge 0 \tag{58}$$

där a är vågutbredningshastigheten. För en entydig lösning till vågekvationen krävs två begynnelsevillkor $u(x,0) = u_0(x), \quad \frac{\partial u}{\partial t}(x,0) = g(x)$ och två randvillkor $u(0,t) = f_0(t), \; u(L,t) = f_L(t)$.

Vi utnyttjar som vanligt finitadifferensmetoden med en indelning av intervallet $0 \le x \le L$ i N delintervall och med steget $h = L/N$ i x-led. Tidssteget är δt och vi önskar beräkna en numerisk lösning till $u(x,t)$ vid tidpunkterna $t_k = k \cdot \delta t$ fram till en slutlig tidpunkt t_{slut}.

Låt u_{ik} beteckna den numeriska approximationen till $u(x_i, t_k)$. Ekvation (58) omformas till

$$\frac{u_{i,k+1} - 2u_{ik} + u_{i,k-1}}{(\delta t)^2} = a^2 \, \frac{u_{i+1,k} - 2u_{ik} + u_{i-1,k}}{h^2}$$

som också kan skrivas

$$u_{i,k+1} = 2u_{ik} - u_{i,k-1} + (\frac{a\,\delta t}{h})^2 (u_{i+1,k} - 2u_{ik} + u_{i-1,k}). \tag{59}$$

Om man känner u-värdena vid tiderna t_{k-1} och t_k är uttrycket (59) en explicit formel för att räkna fram u-värdena vid tiden t_{k+1}. Explicita metoder lider ju alltid av stabilitetstrubbel; för formel (59) kan man visa att stabilitet råder om $a\,\delta t/h \le 1$. Detta kallas för CFL-villkor efter de tre matematikerna Courant, Friedrich, Levi.

Det betyder att om vi har skaffat oss en diskretisering med litet steg h i x-led är vi tvungna att ur stabilitetssynpunkt begränsa tidssteget δt så att det uppfyller $\delta t \le h/a$. Man kan visa att bästa numeriska resultat erhålls då tidssteget tas som rumssteget dividerat med utbredningshastigheten a, alltså $\delta t = h/a$. Eftersom det då gäller att $\frac{a\,\delta t}{h} = 1$ kan formel (59) förenklas till

$$u_{i,k+1} = u_{i-1,k} + u_{i+1,k} - u_{i,k-1}. \tag{60}$$

Vid starten då $k = 0$ kommer (60) att lyda $u_{i,1} = u_{i-1,0} + u_{i+1,0} - u_{i,-1}$. Det måste åtgärdas så att den oönskade termen $u_{i,-1}$ elimineras. De kända uttrycken för u och tidsderivatan vid starten behöver utnyttjas, alltså begynnelsevillkoren $u(x_i, 0) = u_0(x_i)$ och $\frac{\partial u}{\partial t}(x_i, 0) = g(x_i)$.

Vi utnyttjar centraldifferenskvot för tidsderivatan vid $t = 0$ och får följande approximation

$$(u_{i,1} - u_{i,-1})/2\delta t = g(x_i) \quad \Rightarrow \quad u_{i,-1} = u_{i,1} - 2\delta t\, g(x_i).$$

Insättning i formel (60) för $u_{i,1}$ ger $u_{i,1} = u_{i-1,0} + u_{i+1,0} - (u_{i,1} - 2\delta t\, g(x_i))$ som efter lite putsning leder till

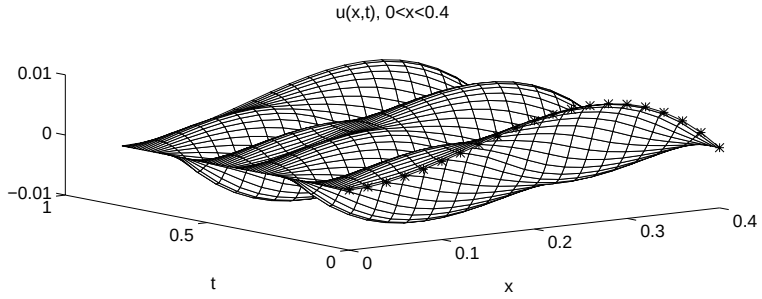
$$u_{i,1} = \frac{u_{i-1,0} + u_{i+1,0}}{2} + \delta t\, g(x_i) \;=\; \frac{u_0(x_{i-1}) + u_0(x_{i+1})}{2} + \delta t\, g(x_i).$$

Med detta uttryck beräknas u-värdet i alla diskretiseringspunkter vid tiden t_1. Därefter fortsätter man beräkningarna med formel (60).

Exempel: Rita vågutbredningen för en sträng med längden $L = 0.4$ och $a = 2$ som är fast inspänd i ändarna, $u(0,t) = u(L,t) = 0$. Strängen släpps från stillastående (tidsderivatan noll) vid läget $u(x,0) = x^2(L - x)$.

Diskretisera strängen i 20 delintervall med rumssteget $h = 0.4/20 = 0.02$. Tidssteget väljs till $\delta t = h/a = 0.02/2 = 0.01$, och beräkningarna utförs fram till tiden 0.8. Strängens läge vid starten är markerad med stjärnor.

```
a=2; L=0.4; N=20; dx=L/N; x=0:dx:L;
dt=dx/a                                                  % bästa val av dt
t=0;  u0=x.^2.*(L-x); u=u0; T=t; U=u';
subplot(2,1,1), plot3(x,zeros(N+1,1),u0,'*'), hold on
g=zeros(N-1,1);                                          % du/dt vid starten
v=[0; (u(1:N-1)+u(3:N+1))/2+dt*g; 0]; t=t+dt; T=[T; t]; U=[U; v'];
while t<0.8
  uold=u; u=v;
  v=[0; u(1:N-1)+u(3:N+1)-uold(2:N); 0]; t=t+dt; T=[T; t]; U=[U; v'];
end
mesh(x',T,U)
```



Studera Heath avsnitt 11.1, 11.2.

11.2 Envägsvågekvationen

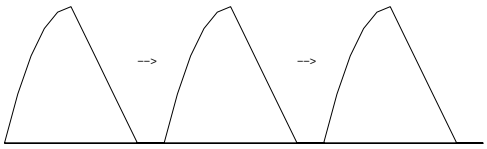
Modellproblemet för en första ordningens hyperbolisk partiell differentialekvation lyder

$$\frac{\partial u}{\partial t} = -a\,\frac{\partial u}{\partial x}, \quad t \geq 0 \tag{61}$$

med begynnelsevillkoret $u(x,0) = f(x)$, där f är ett givet funktionsuttryck som anger vågformen i startögonblicket.

Differentialekvationen satisfieras av $u(x,t) = f(x-at)$ (visas genom derivering och utnyttjande av kedjeregeln).

Det innebär att initialvågen f med oförändrat utseende flyttar sig åt höger (eller åt vänster om $a<0$) med hastigheten a. Lösningen till den linjära modellekvationen (61) bestäms alltså helt av begynnelsevillkoret.



Eventuella diskontinuiteter vid starten finns kvar, ingen spridning eller dämpning sker vilket är fallet i värmeledning och diffusion (paraboliska PDE).

Beteendet är typiskt för hyperboliska PDE; vågor med skarpa vågfronter av typen chockvågor fortplantas oförminskade. På grund av detta sägs differentialekvationen (61) vara konservativ.

Den konserverande egenskapen kan orsaka svårigheter vid numerisk hantering av hyperboliska ekvationer eftersom varje litet numeriskt fel, t ex diskretiseringsfel i finitadifferensmetoden, också konserveras.

11.2.1 Numeriska algoritmer baserade på FDM

För modellekvationen är lösningen $u(x,t)$ helt och hållet känd och ingen numerisk behandling behövs. Men redan vid smärre modifieringar av (61) kan det vara omöjligt att finna en exakt lösning, utan en väl fungerande numerisk metod är nödvändig. Därför är det intressant att studera egenskaperna hos olika numeriska algoritmer även för modellproblemet, där resultatet kan jämföras med den exakta lösningen.

Precis som vid paraboliska och elliptiska ekvationer utnyttjar vi finitadifferensmetoden i lämplig form. I x-led görs diskretisering på vanligt sätt med steget h, och tiden diskretiseras med tidssteget δt.

Den enklaste algoritmen är uppvindsalgoritmen, även kallad FTBS, vilket står för ”Forward-Time Backward-Space”. Den utnyttjar framåtdif-

ferenskvot för tidsderivatan och bakåtdifferenskvot för rumsderivatan, alltså

$$\frac{u_{i,k+1} - u_{i,k}}{\delta t} = -a\,\frac{u_{i,k} - u_{i-1,k}}{h}$$

Om man känner värdena i diskretiseringspunkterna vid tiden t_k får vi följande explicita uttryck för beräkning av u-värdena vid t_{k+1}:

$$u_{i,k+1} = u_{i,k} - \frac{a\,\delta t}{h}(u_{i,k} - u_{i-1,k}). \tag{62}$$

Metoden är stabil om CFL-villkoret $a\,\delta t/h \leq 1$ är uppfyllt, alltså samma krav som i differensuttrycket (59) för vågekvationen. Även här gäller att $a\,\delta t/h = 1$ är ett gott val, så att tidssteget i beräkningarna är $\delta t = h/a$.

I fortsättningen betecknar vi $a\,\delta t/h$ med γ. Formel (62) skrivs alltså $u_{i,k+1} = u_{i,k} - \gamma(u_{i,k} - u_{i-1,k})$. Uppvindsalgoritmen är noggrannhetsmässigt inte särskilt bra, diskretiseringsfelet för bakåtdifferens av rumsderivatan är $O(h)$. I t-led är det Eulers metod som används, och den har som bekant ett fel som är proportionellt mot tidssteget δt. Sammanfattningsvis gäller alltså att uppvindsalgoritmen FTBS ger resultat som är behäftade med ett fel $O(\delta t) + O(h)$.

En annan tänkbar algoritm är FTCS, där CS står för centraldifferenskvot i rummet. Det är en noggrannare derivataapproximation och trunkeringsfelet för rumsdiskretiseringen är $O(h^2)$. Den partiella differentialekvationen (61) approximeras nu av

$$\frac{u_{i,k+1} - u_{i,k}}{\delta t} = -a\,\frac{u_{i+1,k} - u_{i-1,k}}{2h} \quad \Longrightarrow \quad u_{i,k+1} = u_{i,k} - \frac{\gamma}{2}(u_{i+1,k} - u_{i-1,k}).$$

Det visar sig dock att FTCS-algoritmen alltid är instabil och leder till kraftigt växande oscillationer, hur litet δt som än väljs.

11.2.2 Lax-Friedrichs metod och Lax-Wendroffs metod

En modifiering av FTCS är Lax-Friedrichs metod då första högerledstermen $u_{i,k}$ i FTCS ersätts av medelvärdet av omkringliggande termer alltså $(u_{i-1,k} + u_{i+1,k})/2$. Denna lilla förändring visar sig ge stabilitet om $\gamma \leq 1$ är uppfyllt. Formeln för Lax-Friedrichs metod blir

$$u_{i,k+1} = \frac{u_{i-1,k} + u_{i+1,k}}{2} - \frac{\gamma}{2}(u_{i+1,k} - u_{i-1,k}).$$

Med en annan modifiering kan FTCS-algoritmen byggas ut till en stabil och populärare algoritm. Den går under namnet Lax-Wendroffs metod och lyder

$$u_{i,k+1} = u_{i,k} - \frac{\gamma}{2}(u_{i+1,k} - u_{i-1,k}) + \frac{\gamma^2}{2}(u_{i+1,k} - 2u_{i,k} + u_{i-1,k}). \qquad (63)$$

Den tillagda γ^2-termen gör att metoden får samma stabilitetsegenskaper som uppvindsalgoritmen, alltså stabil om $\gamma \leq 1$ är uppfyllt. När det gäller egenskapen noggrannhet är Lax-Wendroffs metod bättre än uppvindsalgoritmen, felet är proportionellt både mot kvadraten på tidssteget och rumssteget, vilket vi skriver $O((\delta t)^2) + O(h^2)$.

Härledning av formel (63) kan göras genom några termer i en taylorutveckling i t-led kring t_k vid diskretiseringspunkten $x = x_i$:

$$u(x_i,\, t_k + \delta t) = u(x_i, t_k) + \delta t \left(\frac{\partial u}{\partial t}\right)_{x_i} + \frac{(\delta t)^2}{2} \left(\frac{\partial^2 u}{\partial t^2}\right)_{x_i} + \ \ldots$$

I taylorutvecklingen ingår tidsderivator som behöver uttryckas som rumsderivator. Här kommer ekvation (61) in, och för förstaderivatan har vi sambandet: $\frac{\partial u}{\partial t} = -a\,\frac{\partial u}{\partial x}$. Derivera båda leden med avseende på t:

$$\frac{\partial^2 u}{\partial t^2} = -a\,\frac{\partial^2 u}{\partial t \partial x} = -a\,\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial t}\right) = -a\,\frac{\partial}{\partial x}\left(-a\frac{\partial u}{\partial x}\right) = a^2\frac{\partial^2 u}{\partial x^2}$$

Sätt in detta i taylorutvecklingen och trunkera efter tre termer:

$$\begin{aligned} u(x_i,\, t_k + \delta t) \ &\approx \ u(x_i, t_k) + \delta t \, \left(\tfrac{\partial u}{\partial t}\right)_{x_i} + \tfrac{(\delta t)^2}{2} \, \left(\tfrac{\partial^2 u}{\partial t^2}\right)_{x_i} = \\ &= \ u(x_i, t_k) - a\,\delta t \, \left(\tfrac{\partial u}{\partial x}\right)_{x_i} + \tfrac{(a\,\delta t)^2}{2} \, \left(\tfrac{\partial^2 u}{\partial x^2}\right)_{x_i} \end{aligned}$$

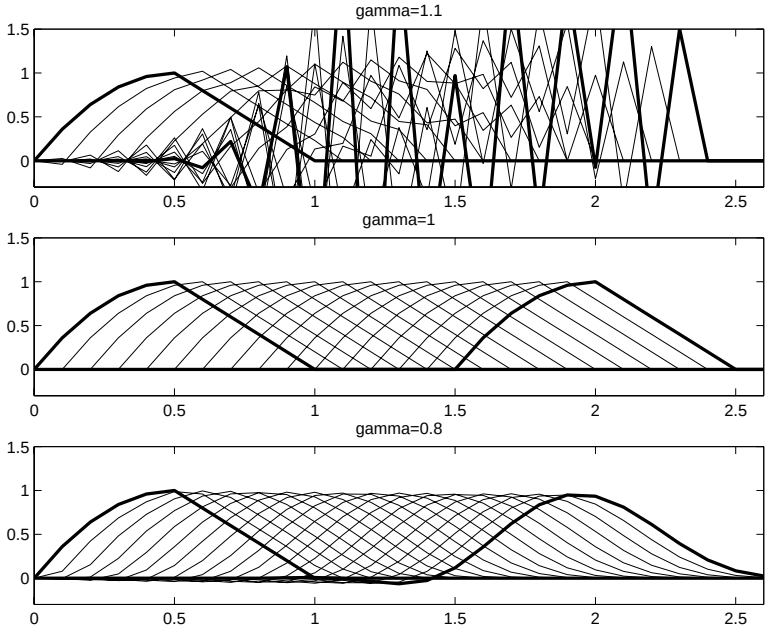
Ersätt rumsderivatorna vid $x = x_i$, $t = t_k$ med differensapproximationerna

$$\left(\frac{\partial u}{\partial x}\right)_{x_i} \approx \frac{1}{2h}(u_{i+1,k} - u_{i-1,k}) \quad \text{och} \quad \left(\frac{\partial^2 u}{\partial x^2}\right)_{x_i} \approx \frac{1}{h^2}(u_{i+1,k} - 2u_{i,k} + u_{i-1,k})$$

båda med ett fel $O(h^2)$. Vid avbrott efter tre termer i taylorutvecklingen erhålls en approximation till $u(x_i, t_{k+1})$ som vi betecknar $u_{i,k+1}$ med det lokala trunkeringsfelet $O((\delta t)^3)$. Det globala felet – då man stegat fram till önskad sluttid t_{slut} – är $O((\delta t)^2)$.

Om vi ersätter första termen i tayloruttryckets högerled med $u_{i,k}$ och byter $a\,\delta t/h$ mot γ, så återfinner vi formel (63). Därmed är härledningen av Lax-Wendroffs algoritm klar.

Exempel: Vi prövar Lax-Wendroffs metod på modellproblemet (61) med $a = 0.5$ och med en vågform vid $t = 0$ som definieras av vektorn `f` i programkoden nedan. Metoden är stabil om $\gamma \leq 1$; med en diskretisering i rumsled på $h = 0.1$ gäller att $\delta t = h\gamma/a \leq 0.1/0.5 = 0.2$. Figurerna visar vad som händer i tre beräkningsfall, först med ett lite för stort tidssteg som ger instabilitet, därefter med idealsteget $\delta t = 0.2$ och slutligen med ett mindre δt som leder till en stabil lösning men där vågformen slätas ut en aning vid varje tidssteg. Vågprofilen vid starten och vid $t_{slut} = 3$ är markerade med tjockare linjer.




```
a=0.5; L=3;
N=30;  h=L/N; x=0:h:L;  x1=0:h:0.5;  x2=0.5+h:h:1;  x3=1+h:h:L;
f=[4*x1.*(1-x1)  2*(1-x2)  zeros(size(x3))];    % vågform vid starten
gamkoll=[1.1  1  0.8];  tslut=3;
for nr=1:3
  gamma=gamkoll(nr); dt=h*gamma/a;  t=0; u=f;
  subplot(3,1,nr), plot(x,f,'LineWidth',2), hold on
  axis([0 2.6 -0.3 1.5]), title(['gamma=' num2str(gamma)])
  while t<tslut
    D2u=u(3:N+1)-2*u(2:N)+u(1:N-1);
    u=[0  u(2:N)-gamma/2*(u(3:N+1)-u(1:N-1))+gamma^2/2*D2u  0];
    t=t+dt; plot(x,u)
  end
  plot(x,u,'LineWidth',2)
end
```

12 Diskret fouriertransform, DFT och FFT

12.1 Anpassning med trigonometriskt polynom

Vi börjar med ett litet exempel för att introducera begreppen trigonometriskt polynom och diskret fouriertransform vid periodiska förlopp.

12.1.1 Trigonometrisk anpassning till sex mätdata

Vi har en uppsättning ekvidistanta mätdata för en periodisk funktion vars period är $T = 3$ sekunder. Data har mätts vid tiderna $t = 0,\ 0.5,\ 1,\ 1.5,\ 2,\ 2.5$ och givit: $y_1 = 2.5,\ y_2 = 5,\ y_3 = 3.5,\ y_4 = 2,\ y_5 = 4$ och $y_6 = 0.5$. De är alltså samplade med tidssteget $\delta t = 0.5$. Vid tiden $t = 3$ kommer det första y-värdet tillbaka igen på grund av det periodiska förloppet. Frekvensen är $f_1 = 1/T$, alltså $1/3$ Hz i detta exempel. Vinkelfrekvensen brukar betecknas ω och definieras av $\omega = 2\pi f_1 = 2\pi/T$.

Vi önskar finna ett trigonometriskt polynom $y(t)$ som interpolerar genom de sex givna punkterna. Grundfrekvensen (grundtonen) är känd, men förutom cosinus- och sinusterm för denna frekvens kan $y(t)$ innehålla övertoner, dvs cosinus- och sinustermer med vinkelfrekvensen 2ω, 3ω, etc.

Eftersom sex mätpunkter är givna kan det trigonometriska polynomet ansättas med sex okända koefficienter:

$$y(t) = a_0 + a_1 \cos\omega t + b_1 \sin\omega t + a_2 \cos 2\omega t + b_2 \sin 2\omega t + a_3 \cos 3\omega t \quad (64)$$

Interpolationskravet ger oss sambanden $F(t_j) = y_j$ för $j = 1, 2, 3, 4, 5, 6$:

$$a_0 + a_1 \cos\omega t_j + b_1 \sin\omega t_j + a_2 \cos 2\omega t_j + b_2 \sin 2\omega t_j + a_3 \cos 3\omega t_j = y_j$$

som i matrisform $\mathbf{Ac} = \mathbf{y}$ kan skrivas

$$\begin{pmatrix} 1 & \cos\omega\,0 & \sin\omega\,0 & \cos 2\omega\,0 & \sin 2\omega\,0 & \cos 3\omega\,0 \\ 1 & \cos\omega\,0.5 & \sin\omega\,0.5 & \cos 2\omega\,0.5 & \sin 2\omega\,0.5 & \cos 3\omega\,0.5 \\ 1 & \cos\omega\,1 & \sin\omega\,1 & \cos 2\omega\,1 & \sin 2\omega\,1 & \cos 3\omega\,1 \\ 1 & \cos\omega\,1.5 & \sin\omega\,1.5 & \cos 2\omega\,1.5 & \sin 2\omega\,1.5 & \cos 3\omega\,1.5 \\ 1 & \cos\omega\,2 & \sin\omega\,2 & \cos 2\omega\,2 & \sin 2\omega\,2 & \cos 3\omega\,2 \\ 1 & \cos\omega\,2.5 & \sin\omega\,2.5 & \cos 2\omega\,2.5 & \sin 2\omega\,2.5 & \cos 3\omega\,2.5 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ b_1 \\ a_2 \\ b_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 2.5 \\ 5.0 \\ 3.5 \\ 2.0 \\ 4.0 \\ 0.5 \end{pmatrix}$$

där a_0, a_1, b_1 a_2, b_2, a_3 är de okända komponenterna i vektorn $\mathbf{c}$. Vinkelfrekvensen ω är $2\pi/3$, och med det värdet insatt i systemmatrisen $\mathbf{A}$ ovan

kommer den att bli

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1/2 & \sqrt{3}/2 & -1/2 & \sqrt{3}/2 & -1 \\ 1 & -1/2 & \sqrt{3}/2 & -1/2 & -\sqrt{3}/2 & 1 \\ 1 & -1 & 0 & 1 & 0 & -1 \\ 1 & -1/2 & -\sqrt{3}/2 & -1/2 & \sqrt{3}/2 & 1 \\ 1 & 1/2 & -\sqrt{3}/2 & -1/2 & -\sqrt{3}/2 & -1 \end{pmatrix}$$

Man kan undra varför högsta frekvensen 3ω i uttrycket (64) inte har någon sinusterm utan bara en $\cos 3\omega t$. Ja, ta och beräkna $\sin 3\omega t_j$ för de sex givna t_j-värdena och avgör vilken information som dessa värden skulle kunna tänkas tillföra!

Med sex data är det trigonometriska polynomet (64) uppbyggt av termer med grundfrekvensen $f_1 = \frac{1}{T}$ samt de två övertonerna $f_2 = \frac{2}{T}$ och $f_3 = \frac{3}{T}$. Högre frekvenser är inte möjliga att urskilja vid denna sampling.

Den maximala frekvensen kallas nyquistfrekvensen och den bestäms av $f_{max} = \frac{N/2}{T}$, när N är antalet samplade data under en period. Nyquistfrekvensen är alltså i vårt exempel $\frac{3}{3} = 1$.

Ett naturligt sätt att erhålla de obekanta koefficienterna skulle vara att lösa systemet $\mathbf{Ac} = \mathbf{y}$ med gausselimination. Men så går man aldrig tillväga! Man utnyttjar i stället att matrisen vid ekvidistant samplade mätdata har en trevlig ortogonalitetsegenskap: kolumnerna i $\mathbf{A}$ är ortogonala mot varandra. Multiplicera vänsterled och högerled i ekvationssystemet med $\mathbf{A}^T$ (precis som vid minstakvadratmetoden), alltså $\mathbf{A}^T\mathbf{Ac} = \mathbf{A}^T\mathbf{y}$.

Matrisen $\mathbf{A}^T\mathbf{A}$ blir på grund av ortogonaliteten en diagonalmatris, och i fallet med sex mätdata blir diagonalelementen 6, 3, 3, 3, 3, 6 (förvissa dig om detta). Då antalet mätdata är N kommer första och sista diagonalelementet att vara N och de övriga $N/2$.

Högerledsvektorn $\mathbf{A}^T\mathbf{y}$ kommer i vårt fall att innehålla sex komponenter som i tur och ordning är

$$\begin{gathered} \textstyle\sum_{j=1}^{6} y_j, \quad \sum_{j=1}^{6} y_j \cos \omega t_j, \quad \sum_{j=1}^{6} y_j \sin \omega t_j, \\ \textstyle\sum_{j=1}^{6} y_j \cos 2\omega t_j, \quad \sum_{j=1}^{6} y_j \sin 2\omega t_j, \quad \sum_{j=1}^{6} y_j \cos 3\omega t_j. \end{gathered}$$

Eftersom ekvationssystemet nu fått en diagonalmatris, erhålls varje lösningskomponent som högerledselementet dividerat med diagonalelementet, alltså

$$\begin{gathered} a_0 = \tfrac{1}{6}\textstyle\sum_{j=1}^{6} y_j, \;\; a_1 = \tfrac{1}{3}\sum_{j=1}^{6} y_j \cos \omega t_j, \;\; b_1 = \tfrac{1}{3}\sum_{j=1}^{6} y_j \sin \omega t_j, \\ a_2 = \tfrac{1}{3}\textstyle\sum_{j=1}^{6} y_j \cos 2\omega t_j, \; b_2 = \tfrac{1}{3}\sum_{j=1}^{6} y_j \sin 2\omega t_j, \; a_3 = \tfrac{1}{6}\sum_{j=1}^{6} y_j \cos 3\omega t_j. \end{gathered}$$

I vårt exempel får fourierkoefficienterna a_k och b_k följande värden:
$a_0 = 2.9167$, $a_1 = -0.1667$, $b_1 = 1.1547$, $a_2 = -0.6667$, $b_2 = 1.4434$, $a_3 = 0.4167$.

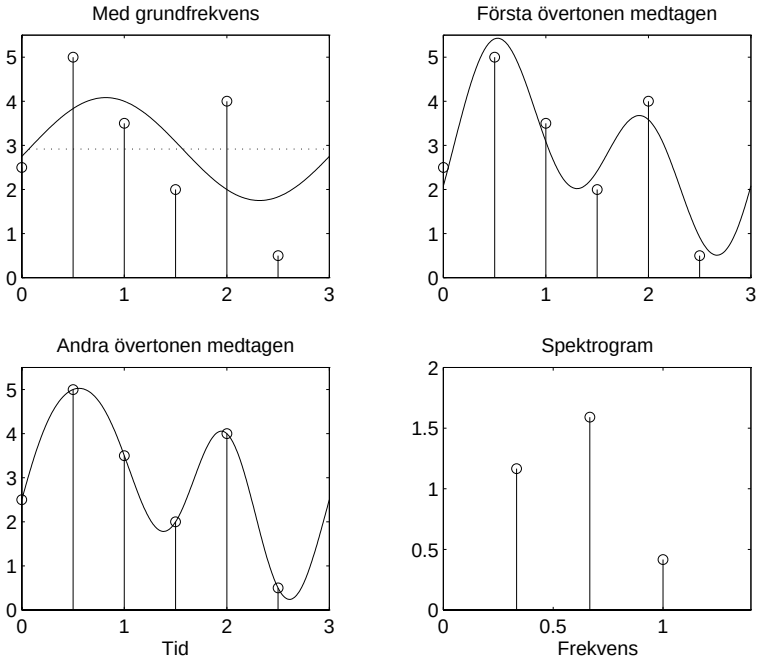
Första koefficienten a_0 är medelvärdet av mätdata. De övriga termerna i det trigonometriska polynomet (64) kan vi para ihop:

$$y(t) = a_0 + (a_1 \cos \omega t + b_1 \sin \omega t) + (a_2 \cos 2\omega t + b_2 \sin 2\omega t) + a_3 \cos 3\omega t$$

och de utgörs av tre sinussvängningar vars amplituder bestäms av

$$\sqrt{a_1^2 + b_1^2} = 1.1667, \quad \sqrt{a_2^2 + b_2^2} = 1.5899, \quad |a_3| = 0.4167.$$

I ett spektrogram åskådliggör man amplituden vid varje frekvenskomponent.



Figuren visar hur väl data approximeras först av enbart grundtonen (de tre första termerna i det trigonometriska polynomet), därefter med första övertonen tillagd (fem termer medtagna) och till slut den interpolerande kurvan med alla sex termerna medtagna. I nedre högra bilden visas spektrogrammet.

Studera Heath avsnitt 12.1, 12.2, 12.3, 12.6.

12.1.2 DFT och fourierkoefficienter, allmänt

När man har N stycken ekvidistant samplade mätdata över en period T, kommer det trigonometriska polynomet (64) att utvidgas till

$$y(t) = a_0 + a_1 \cos \omega t + b_1 \sin \omega t + a_2 \cos 2\omega t + b_2 \sin 2\omega t + \cdots + a_m \cos m\omega t \quad (65)$$

där $m = N/2$ och $\omega = 2\pi/T$. De totalt N stycken fourierkoefficienterna bestäms av uttrycken:

$$\begin{gathered} a_0 = \tfrac{1}{N} \textstyle\sum_{j=1}^{N} y_j, \\ a_k = \tfrac{1}{m} \textstyle\sum_{j=1}^{N} y_j \cos k\omega t_j, \quad b_k = \tfrac{1}{m} \textstyle\sum_{j=1}^{N} y_j \sin k\omega t_j, \quad k = 1, 2, ..., m-1 \\ a_m = \tfrac{1}{N} \textstyle\sum_{j=1}^{N} y_j \cos m\omega t_j \end{gathered} \quad (66)$$

Detta förfarande kallas diskret fouriertransform, DFT.

12.2 Komplex ansats — DFT fiffigt och effektivt med FFT

Med hjälp av Eulers identitet $e^{i\varphi} = \cos\varphi + i \sin\varphi$ kan det trigonometriska uttrycket $y(t)$ skrivas med en komplex ansats. I fallet med sex samplade data kan följande sex ekvationer $(j = 1, 2, ..., 6)$ ställas upp:

$$\frac{1}{6}\left(Y_0 + Y_1 e^{i\omega t_j} + Y_2 e^{2i\omega t_j} + Y_3 e^{3i\omega t_j} + Y_4 e^{4i\omega t_j} + Y_5 e^{5i\omega t_j}\right) = y_j \quad (67)$$

för bestämning av sex okända komplexa koefficienter Y_0, $Y_1, \ldots, Y_5$.
I formel (64) ingick vinkelfrekvenserna ω, 2ω, 3ω, men i det komplexa uttrycket (67) tycks det som om vinkelfrekvenserna 4ω och 5ω också finns med. Detta är faktiskt en ren beräkningsfiness — egentligen ser ansatsen för det komplexa uttrycket ut så här:

$$\frac{1}{6}\left[Y_0 + (Y_1 e^{i\omega t_j} + Y_{-1} e^{-i\omega t_j}) + (Y_2 e^{2i\omega t_j} + Y_{-2} e^{-2i\omega t_j}) + Y_3 e^{3i\omega t_j}\right] = y_j$$

Man kan dock visa att för de $N = 6$ samplingspunkterna t_j gäller identiskt $e^{-i\omega t_j} = e^{(N-1)i\omega t_j}$ och $e^{-2i\omega t_j} = e^{(N-2)i\omega t_j}$. Detta leder till den övre formeln (67) om vi samtidigt döper om Y_{-1} till $Y_{N-1} = Y_5$ och Y_{-2} till $Y_{N-2} = Y_4$.
Allmänt gäller att den komplexa fourieransatsen för N samplade data $(t_j,\, y_j)$ skrivs på formen

$$y_j = \frac{1}{N} \sum_{k=0}^{N-1} Y_k e^{ik\omega t_j}, \quad j = 1, 2, \ldots, N \quad (68)$$

där de komplexa fourierkoefficienterna Y_k bestäms av

$$Y_k = \sum_{j=1}^{N} y_j e^{-ik\omega t_j},\ k = 0, 1, 2, \ldots, N-1. \tag{69}$$

Termen Y_0/N är medelvärdet av mätdata, alltså lika med fourierkoefficienten a_0 i (66). Liksom tidigare låter vi m beteckna $N/2$. Då gäller att Y_m/N är koefficienten för nyquistfrekvenstermen, Y_m/N överensstämmer med a_m i formel (66). Övriga samband är:

$$Y_k/m = a_k - i\,b_k, \quad Y_{N-k}/m = a_k + i\,b_k \ \text{ för } \ k = 1, 2, \ldots, m-1.$$

Det innebär alltså att Y_{N-k} är komplexkonjugatet av Y_k, därför räcker det att beräkna m koefficienter.

Omvänt kan fourierkoefficienterna a_k och b_k för $k = 1, 2, \ldots, m-1$ uttryckas i de komplexa koefficienterna enligt

$$a_k = \mathrm{Re}(Y_k)/m, \quad b_k = -\mathrm{Im}(Y_k)/m.$$

För den uppsättning data som vi hade i avsnitt 12.1.1, alltså sex y_j-värden 2.5, 5, 3.5, 2, 4.0, 0.5, uppmätta vid $t_j = 0$, 0.5, 1, 1.5, 2, 2.5 erhålls de komplexa fourierkoefficienterna:

$$\begin{array}{lll} Y_0 = 17.5, & Y_1 = -0.5 - 3.4641i, & Y_2 = -2.0 - 4.3301i, \\ Y_3 = 2.5, & Y_4 = -2.0 + 4.3301i, & Y_5 = -0.5 + 3.4641i. \end{array}$$

Kontrollera att de ovan nämnda fourierkoefficientsambanden stämmer i detta exempel!

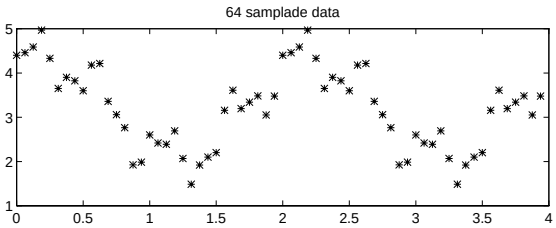
Antalet operationer för att bestämma de komplexa fourierkoefficienterna Y_k är proportionellt mot N^2 vid summation enligt formel (69). Men genom listighet vid summeringen kan Y_k beräknas effektivare med den så kallade snabba fouriertransformen, *Fast Fourier Transform*[8], förkortat FFT, som i MATLAB finns som standardfunktionen `fft`. Antalet operationer blir i FFT-algoritmen proportionellt mot $N \log N$. Antalet samplingspunkter N väljs alltid som ett jämnt tal, oftast som en tvåpotens.

[8]För härledning av FFT-algoritmen, se Heath, avsnitt 12.2.

12.3 Tillämpningar med DFT och FFT

12.3.1 Diskret fouriertransform till 64 mätdata

Bestäm DFT för figurens 64 data samplade under en tidsrymd på $T = 4$ sekunder vid tidpunkterna $t_j = 0,\ \delta t,\ 2\delta t, \ldots, 63\delta t$, där $\delta t = T/64$.



Bestäm nyquistfrekvensen och beräkna fourierkoefficienterna a_k och b_k dels med de explicita uttrycken i (66), dels med hjälp av MATLABs `fft`.

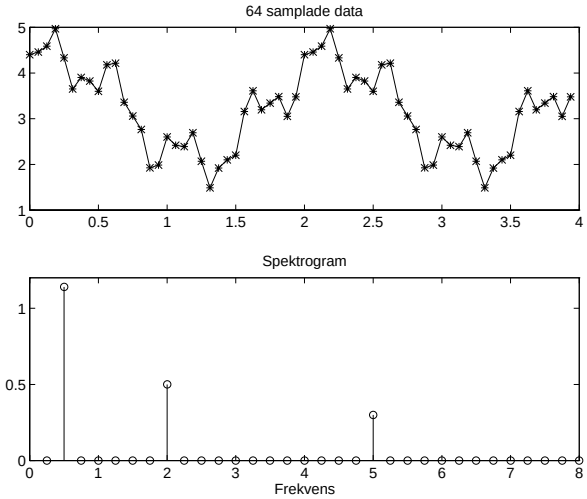
Eftersom de samplade datapunkterna i själva verket skapats ur följande uttryck av cosinus- och sinustermer

$$y(t) = 3.2 + 0.9\cos \pi t + 0.7\sin \pi t + 0.5\sin 4\pi t + 0.3\cos 10\pi t$$

bör spektrogrammet för den beräknade diskreta fouriertransformen återge just uttryckets tre frekvenser.

Nyquistfrekvensen bestäms av $f_{max} = \frac{N/2}{T} = \frac{32}{4}$ och är alltså 8 Hz. Ett förslag till programkod är följande:

```
% dftex2,  diskret fouriertransform med explicita formler och med FFT
  clear, clf
  N=64; T=4; dt=T/N; t=dt*(0:N-1)'; m=N/2; df=1/T;
  y=3.2+0.9*cos(pi*t)+0.7*sin(pi*t)+0.5*sin(4*pi*t)+0.3*cos(10*pi*t);
  subplot(2,1,1), plot(t,y,'*'), title('64 samplade data'), hold on
  tt=2*pi*t;
  a0=sum(y)/N;                              % a0 är medelvärdet av data
  Af=[]; Bf=[];
  for k=1:m-1, wt=k*tt/T;                   % övriga fourierkoeff ak och bk
    ak=sum(y.*cos(wt))/m; Af=[Af; ak];
    bk=sum(y.*sin(wt))/m; Bf=[Bf; bk];
  end
  am=sum(y.*cos(m*tt/T))/N; Af=[Af; am]; Bf=[Bf; 0];
  Y=fft(y);                                 % fourierkoeff med FFT
  A0=Y(1)/N;                                % fourierkoefficienten A0=a0
  Y(1)=[];   C=Y(1:m)/m; C(m)=C(m)/2;       % övriga koeff (komplex form)
  A=real(C); B=-imag(C);                    % koeff för cos- och sintermer
  yapp=A0;
  for k=1:m, wt=k*tt/T; yapp=yapp+A(k)*cos(wt)+B(k)*sin(wt); end
  plot(t,yapp,'c')
  freq=df*(1:m)';
  subplot(2,1,2), stem(freq,abs(C)), title('Spektrogram'), xlabel('Frekvens')
% Kolla att beräkningsmetoderna ger samma fourierkoefficienter
  Adiffkoll=norm(Af-A,inf), Bdiffkoll=norm(Bf-B,inf)
% Resultat: Adiffkoll=7e-15, Bdiffkoll=6e-15
```



Spektrogrammet visar att de tre frekvenserna 0.5 Hz, 2 Hz och 5 Hz finns i den diskreta fouriertransformen med amplituderna 1.1402, 0.5000 och 0.3000, vilket var väntat (amplitudvärdet 1.1402 är $\sqrt{0.9^2 + 0.7^2}$).

12.3.2 Frekvensanalys av brusade data

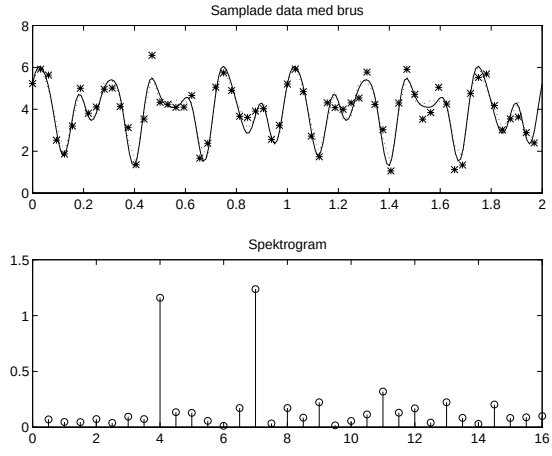
Bestäm nu den diskreta fouriertransformen (DFT) för 64 samplade punkter under tidsperioden två sekunder för funktionen

$$y(t) = 4 + 0.9\cos(2\pi f_1 t) + 0.7\sin(2\pi f_1 t) + 1.2\sin(2\pi f_2 t) + 0.5\cos(2\pi f_3 t)$$

med $f_1 = 4$ Hz, $f_2 = 7$ Hz och $f_3 = 11$ Hz. Vi lägger på ett normalfördelat brus med standardavvikelsen 0.5 och medelvärdet noll. De brusade mätpunkterna är asteriskmärkta och den ursprungliga ostörda funktionen är prickad i figuren. Programmet utför FFT och skapar ett spektrogram.

På grund av bruset kommer spektrum inte bara att innehålla de tre ursprungliga frekvenserna utan det blir bidrag från nästan alla frekvenser från den lägsta $1/T = 0.5$ Hz till nyquistfrekvensen $N/(2T) = 16$ Hz.

Vi låter programmet plocka ut de fourierkoefficienter som överstiger tjugo procent av maximala fourierkoefficienten, resten filtreras bort.



I detta fall kommer vektorn `Frekv` att innehålla de tre komponenterna 4, 7 och 11. Den heldragna kurvan visar den approximerande fourierutvecklingen med dessa tre medtagna frekvenser.

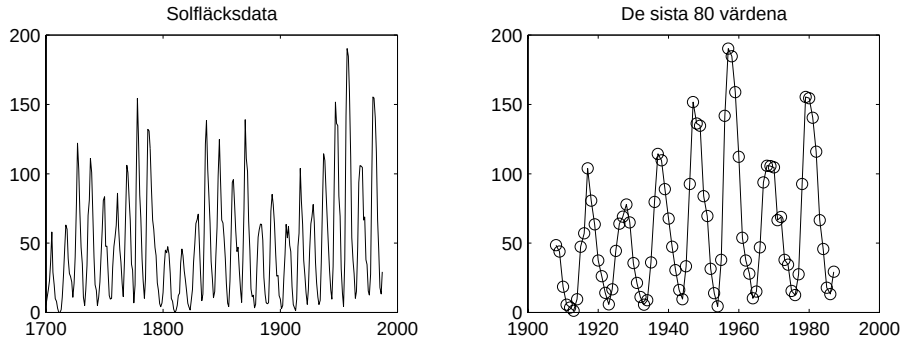
```
N=64; T=2; dt=T/N; t=dt*(0:N-1)'; m=N/2; df=1/T;
f1=4; f2=7; f3=11; tt=2*pi*t;
y=4+0.9*cos(f1*tt)+0.7*sin(f1*tt)+1.2*sin(f2*tt)+0.5*cos(f3*tt);
yb=y+0.5*randn(N,1);                  % data med normalfördelat brus
subplot(2,1,1), plot(t,yb,'*', t,y,':'), hold on
Y=fft(yb);  A0=Y(1)/N;
Y(1)=[]; C=Y(1:m)/m; C(m)=C(m)/2;
A=real(C); B=-imag(C);  freq=df*(1:m)';
subplot(2,1,2), stem(freq,abs(C)), title('Spektrogram')
Cmax=max(abs(C)); reltol=0.2;         % tjugoprocentsgräns
nr=find(abs(C)>reltol*Cmax);          % hitta index till stora koeff
Frekv=df*nr'                          % frekvenser som tas med
tx=0:0.01:2; yapp=A0;                 % tät tidsindelning för ritningen
for k=1:length(nr)
  wt=2*pi*nr(k)*tx/T;                 % summera termer med stora koeff
  yapp=yapp+A(nr(k))*cos(wt)+B(nr(k))*sin(wt);
end
subplot(2,1,1), plot(tx,yapp,'c')
```

12.4 Solfläcksaktivitet

Denna tillämpning av fourierteknik som gäller undersökning av periodiskt förekommande solfläcksaktivitet kan vi finna i boken av *Kahaner&Moler&Nash, Numerical Methods and Software, Problem P11-12.*

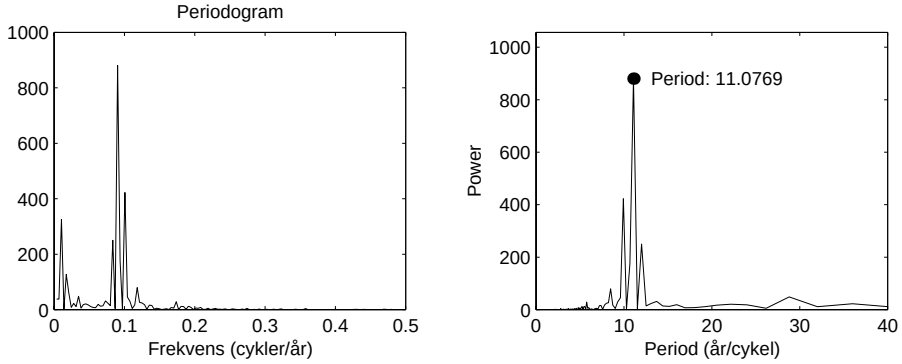
For centuries people have noted that the face of the sun is not constant or uniform in appearance, but that darker regions appear at random locations on a cyclical basis. In 1848 Rudolf Wolfer proposed a rule that combined the number and size of these dark spots into a single number. Using archival records astronomers have applied Wolfer's rule and determined sunspot numbers back to the year 1700. Today these are measured by many observatories and the worldwide distribution of the data is coordinated by the Swiss Federal Observatory on a daily, monthly and yearly basis. Sunspot activity is cyclical and variation in the Wolfer numbers has been correlated with weather and terrestrial phenomena of economic significance; this accounts for the continuing interest in them.

The file `sunspot.dat` contains the average Wolfer sunspot number for each year from 1700 to 1987. Write a program to read these numbers and plot them. Then use FFT on this data. The numbers do not have any noticable trend that needs to be removed. Plot the power as a function of frequency (the power component P_k is the square of the absolute value of the k-th complex fourier coefficient). Is there a single frequency that dominates? What is the period of the dominant cycle? Between 1987 and the turn of the next century, in what years will sunspot activity be a maximum?



Till vänster visas samtliga 288 data, och till höger de sista 80 solfläckstalen (1908 - 1987). Vi gör FFT och ritar upp ett periodogram ("power spectrum"), som visar att det finns en dominerande fourierkoefficient, nämligen nummer 26 (variabeln `ip`) med frekvensen $26/288 = 0.0903$ (i enheten $år^{-1}$). Om vi inverterar frekvensen erhålls periodtiden $1/0.0903 = 11.08$ år. Solfläcksdata sträcker sig fram till år 1987, och 1979 hittar vi sista maximivärdet.

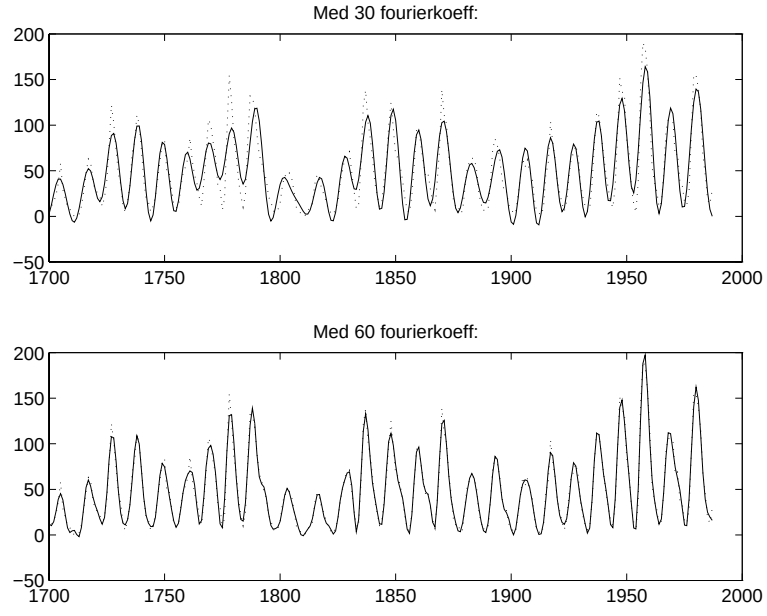
Elva år senare, alltså 1990 och därefter år 2001 har man mycket riktigt kunnat konstatera avsevärd solfläcksaktivitet.




```
% solflackfft, FFT på solfläcksdata från år 1700 till 1987
  clear, figure(1), clf
  load sunspot.dat
  tid=sunspot(:,1);  ys=sunspot(:,2); N=length(tid); m=N/2;
  subplot(2,2,1), plot(tid,ys), title('Solfläcksdata'), drawnow
  t80=tid(N-80+1:N); y80=ys(N-80+1:N);
  subplot(2,2,2), plot(t80,y80, t80,y80,'mo'), title('De sista 80 värdena')
  G=fft(ys);
  yapp=G(1)/N;  G(1)=[];  C=G(1:m)/m;
  dt=1; df=1/(N*dt), freq=df*(1:m);
  P=abs(C).^2;
  subplot(2,2,3), plot(freq,P), xlabel('Frekvens (cykler/år)')
  title('Periodogram'), drawnow
```

```
period=1./freq;                % byt till inverterad skalning: år/cykel
subplot(2,2,4), plot(period,P)
xlabel('Period (år/cykel)'), ylabel('Power'), hold on
ip=find(P==max(P)), pcyk=period(ip), powcyk=P(ip);
plot(pcyk,powcyk,'r.','MarkerSize',20)
text(pcyk+2,powcyk,['Period: ', num2str(pcyk)])
axis([0 40 0 1.2*powcyk]), drawnow
A=real(C); B=-imag(C);
t=tid-tid(1);                          % transformera så att t(1)=0
for j=1:ip+3
  wt=2*pi*j*df*t;  yapp=yapp+A(j)*cos(wt)+B(j)*sin(wt);
end
figure(2), clf
subplot(2,1,1), plot(tid,ys,':', tid,yapp)
title(['Med ' int2str(ip+4) ' fourierkoeff:'])
for j=ip+4:60                          % 60 koefficienter medtagna
  wt=2*pi*j*df*t;  yapp=yapp+A(j)*cos(wt)+B(j)*sin(wt);
end
subplot(2,1,2), plot(tid,ys,':', tid,yapp), title('Med 60 fourierkoeff:')
```

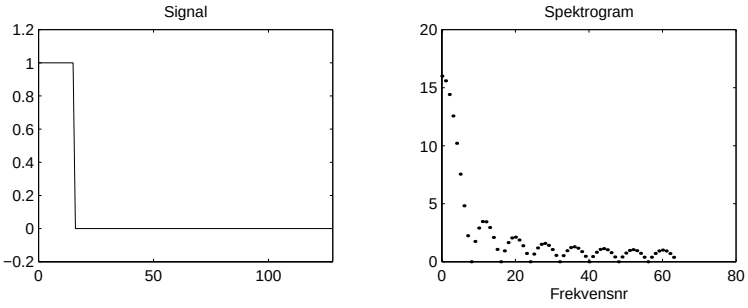
I följande figur är solfläcksdata inprickade och den heldragna kurvan är det approximerande trigonometriska polynomet med de 30 första frekvenserna. I den undre figuren har de 60 första frekvenserna tagits med.



12.5 Fouriertransform för en kortvarig puls

Rita ett spektrogram för en puls som har konstant amplitud under en åttondel av en hel samplingsperiod som omfattar 128 data. Våra y-värden består av en vektor med $N = 128$, vars första 16 komponenter är ett, övriga är noll.

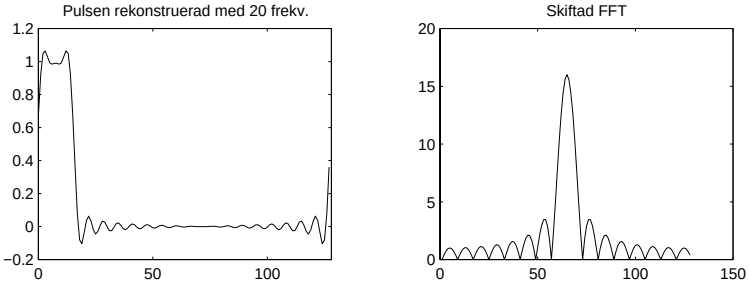
Spektrogrammet visar att de största fourierkoefficientamplituderna finns vid låga frekvenser och att spektrum har ett $\frac{\sin x}{x}$-beteende.




```
N=128;  y=zeros(N,1); y(1:16)=1;
subplot(2,2,1), plot(0:127,y), title('Signal'), axis([0 N -0.2 1.2])
Y=fft(y);
subplot(2,2,2), plot(0:63,abs(Y(1:64)),'.')
xlabel('Frekvensnr'), title('Spektrogram')
```

I exemplet har vi låtit diskretiseringssteget δt vara ett, så att perioden som vi kallat T tidigare överensstämmer med antalet samplade punkter N. Det förenklar programkoden, och det gör också följande: I spektrogrammet borde egentligen amplitudvärdena ha en faktor $m = N/2$ i nämnaren, men vilken skala som spektrogrammet har på vertikala axeln är mestadels ointressant, alltså skalar vi bort den!

Nedan ser vi en rekonstruktion av pulsen då enbart 20 av de 64 ingående frekvenserna utnyttjas. Som synes uppstår små oscillationer vid pulsens kanter. Detta går under benämningen Gibbs fenomen. Vad kan högra bilden med rubriken "Skiftad FFT" tänkas föreställa?



```
Ytrunk=Y;  Ytrunk(21:N-20)=0;  yapp=ifft(Ytrunk);
subplot(2,2,3), plot(0:127,yapp), axis([0 N -0.2 1.2])
title('Pulsen rekonstruerad med 20 frekv.')
Yg=fftshift(Y); subplot(2,2,4), plot(1:N,abs(Yg)), title('Skiftad FFT')
```

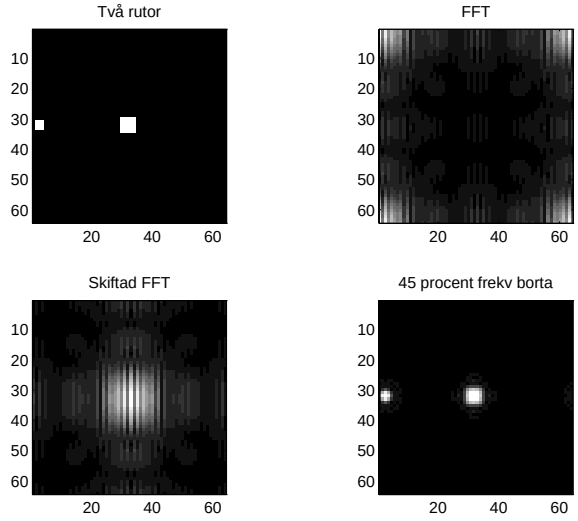
Med `fftshift` på fourierkoefficientvektorn kan man tillverka ett ”skiftat” spektrogram, mer om det i nästa avsnitt.

Sammanfattning: Diskret fouriertransform kan utföras på en vektor $\mathbf{y}$ med N komponenter. Den komplexa fouriertransformen erhålls med hjälp av `Y=fft(y)`, vilket ger en vektor $\mathbf{Y}$ med N komplexa komponenter. Vi vet att det skulle räcka med hälften eftersom Y_{N-k} är komplexkonjugatet av Y_k (och $|Y_{N-k}| = |Y_k|$). Men man behåller gärna alla N komponenterna i fourierkoefficientvektorn så att $\mathbf{y}$ och $\mathbf{Y}$ har lika många element.

När steget ska tas från endimensionell till tvådimensionell fouriertransform har vi nytta av vektorbetraktelsen här. Men i 2D-fallet blir det inte vektorer utan matriser som ska utsättas för diskret fouriertransform med hjälp av FFT-algoritmer.

12.6 FFT i två dimensioner på en enkel bildmatris

Vi ska studera fouriertransformen för en 64×64 bildmatris som innehåller två små vita rutor, en med 3×3 element och en med 5×5, enligt vänstra figuren. Här kan vi tillämpa MATLABs tvådimensionella fouriertransform `fft2`, som egentligen består av vanlig endimensionell FFT utförd i tur och ordning på de 64 raderna i bildmatrisen, följd av FFT på de just erhållna matrisvärdenas kolumner från den första till den sextiofjärde kolumnen.



Den högra figuren visar en intensitetsbild av absolutbeloppen för de komplexa fourierkoefficienterna — ju större belopp, desto ljusare är motsvarande matriselement. Det räcker fullständigt att betrakta en kvarts fourierkoefficientmatris, t ex övre vänstra fjärdedelen av matrisen. Spegling kring vertikala mittaxeln ger högersidan; på samma sätt får vi den undre matrisdelen genom spegling i horisontella mittaxeln.

Vanligtvis åskådliggörs 2D-fouriertransformen så som den undre vänstra figuren visar, alltså så att största ljusintensiteten hamnar i mitten. Man vänder helt enkelt upp och ned och byter vänster–höger på varje fjärdedel av fourierkoefficientmatrisen. I MATLAB görs det med `fftshift`. (Denna symmetriska FFT-figur anses vara vackrare än bara den nödvändiga och tillräckliga fjärdedelen av den.)

Det är intressant att undersöka hur den ursprungliga bildens vita kvadrater deformeras och mer eller mindre suddas ut då höga frekvenser plockas bort. I den högra figuren är de 45 procent högsta frekvenserna är bortkapade. Programkoden innehåller denna möjlighet till frekvensbegränsning.

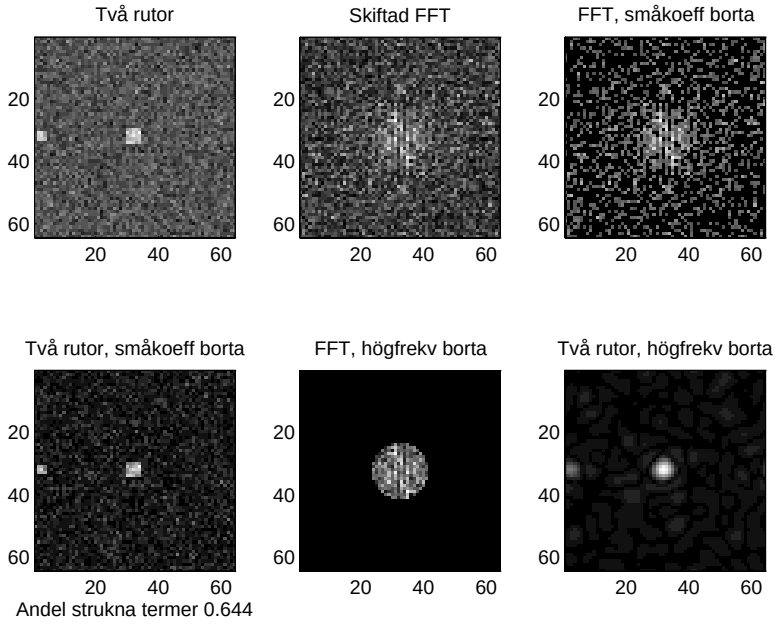
```
% fftkvadrat, 2D-fouriertransform på två kvadrater
  colormap(gray(16))                                        % gråskala
  N=64; m=N/2;
  z=zeros(N); z(30:34,30:34)=1; z(31:33,2:4)=1;
  subplot(2,2,1), imagesc(z), title('Två rutor'), axis image
  G=fft2(z); subplot(2,2,2), imagesc(abs(G)), title('FFT'), axis image
  Gg=fftshift(G);
  subplot(2,2,3), imagesc(abs(Gg)), title('Skiftad FFT'), axis image
  k=1:m; [X,Y]=meshgrid(k,k);
  disp('Andel medtagna frekvenser (bryt med p=0).')
  while 1==1
    p=input('p: '); if p>=1 | p<=0, break, end
    r=p*m;  Hq=X.^2+Y.^2<r^2; H=[Hq fliplr(Hq)]; H=[H; flipud(H)];
    Gsm=H.*G;  zapp=ifft2(Gsm);
    subplot(2,2,4), imagesc(abs(zapp)), axis image
    title([int2str(100*(1-p)) ' procent frekv borta'])
  end
```

12.6.1 Bildmatris med brus

Vi lägger på ett normalfördelat brus med standardavvikelse 0.2 och medelvärdet noll på elementen i bildmatrisen, så att den får utseendet enligt övre vänstra bilden i nästa bildsekvens. Efter att ha utfört `fft2` kan vi antingen plocka bort till beloppet små fourierkoefficienter, t ex de som är mindre än 30 % av maxkoefficienten, eller kapa de högsta frekvenserna. I programkoden prövas båda möjligheterna.

När de 30 procent minsta koefficienterna sätts till noll försvinner faktiskt 64 procent av termerna i fourierutvecklingen. Rekonstruktionen av den brusade bilden blir som nedre vänstra figuren visar.

Om man i stället väljer att kapa de 70 procent högsta frekvenserna blir rekonstruktionen av bildmatrisen annorlunda, resultatet ses längst till höger.



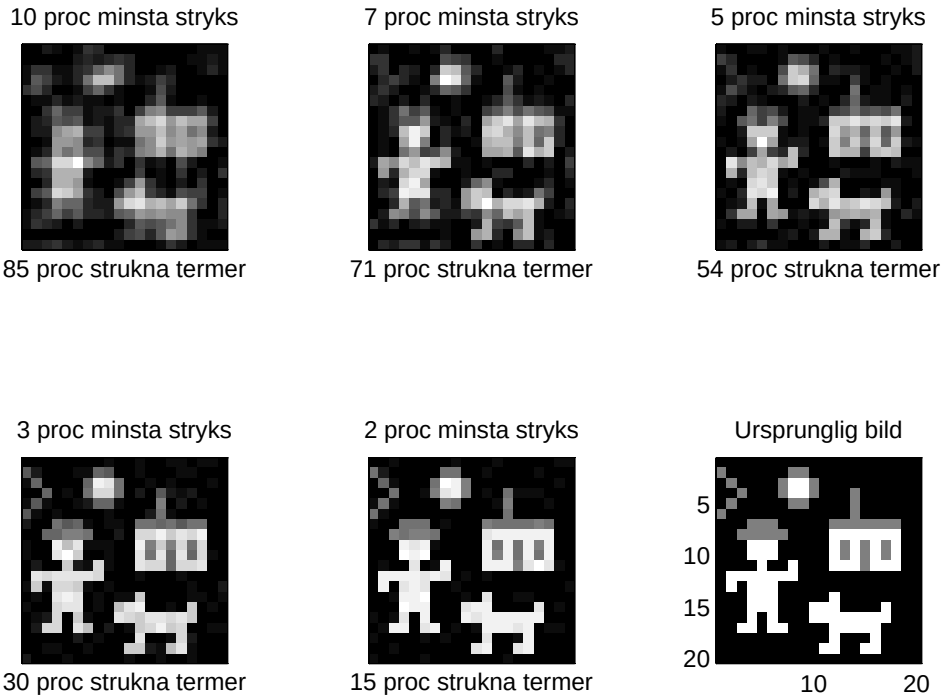

```
% fftkvadratbrus, 2D-fouriertransform på bildmatris med brus
  clear, clf
  colormap(gray(16)), N=64; df=1/N; m=N/2;
  z=zeros(N); z(30:34,30:34)=1; z(31:33,2:4)=1;  z=z+0.2*randn(N);
  subplot(2,3,1), imagesc(z), title('Två rutor'), axis image
  G=fft2(z); Gg=fftshift(G);
  subplot(2,3,2), imagesc(abs(Gg)), title('Skiftad FFT'), axis image

  Gmax=max(max(abs(G)));  p=0.3              % stryk koeff mindre än 0.3*Gmax
  H=abs(G/Gmax)>0.3; [ih,jh]=find(H==0); andelstrukna=length(ih)/N^2
  Gsm=H.*G; Ggsm=fftshift(Gsm);
  subplot(2,3,3), imagesc(abs(Ggsm)), title('FFT,småkoeff borta'), axis image
  zapp=ifft2(Gsm);
  subplot(2,3,4), imagesc(abs(zapp)), axis image
  title(['Två rutor, småkoeff borta'])
  xlabel(['Andel strukna termer ' num2str(andelstrukna)])

  k=1:m; [X,Y]=meshgrid(k,k);
  p=0.3; r=p*m;  Hq=X.^2+Y.^2<r^2; H=[Hq fliplr(Hq)]; H=[H; flipud(H)];
  Gsm=H.*G;  Ggsm=fftshift(Gsm);
```

```
subplot(2,3,5), imagesc(abs(Ggsm)), title('FFT, högfrekv borta'), axis image
zapp=ifft2(Gsm);
subplot(2,3,6), imagesc(abs(zapp)), axis image
title('Två rutor, högfrekv borta')
```

12.7 Bildmatris behandlad dels med FFT dels med SVD

Vi skapar ett något intressantare bildinnehåll än bara två små kvadrater. Vad sägs om en bild på en hund med husse, hus och stiliserad sol. Bildmatrisen är inte stor, den har bara 20×20 element.

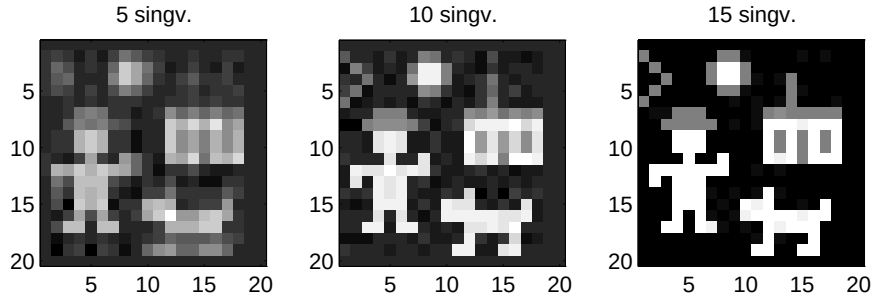
På denna matris utför vi FFT på samma sätt som i förra exemplet. Vi plockar sedan bort fouriertermer med små koefficienter. Om de tio procent minsta koefficienterna tas bort, försvinner 85 procent av termerna. Bildkvaliteten är rätt dålig vilket första figuren visar. Följ bildsekvensen och konstatera hur bildkvaliteten förbättras i takt med att fourierkoefficienter med allt mindre belopp tas med.



```
% hundhussefft, datakomprimering med FFT, stryk små fourierkoefficienter
  clear, clf
  colormap(gray(21)), load Hundhusse, A=Hundhusse;  mn=size(A);
  G=fft2(A); Gmax=max(max(abs(G)));  R=[10 7 5 3 2];
  for nr=1:5                                  % stryk termer med små koeff
    H=abs(G/Gmax)>R(nr)*0.01;
    [ih,jh]=find(H==0); andelstrukna=length(ih)/prod(mn)
    Gsm=H.*G; Aapp=ifft2(Gsm);
    subplot(2,3,nr), imagesc(abs(Aapp)), axis image
    title([int2str(R(nr)) ' proc minsta stryks'])
    set(gca,'YTick',[],'XTick',[])
    xlabel([int2str(100*andelstrukna) ' proc strukna termer']), drawnow
  end
  subplot(2,3,6), imagesc(A), axis image, title('Ursprunglig bild')
```

Nu prövar vi trunkerad singulärvärdesfaktorisering på samma hund–husse–hus–matris. Figurerna nedan har först fem medtagna singulärvärden, därefter tio och femton. Som synes är resultaten snarlika i FFT- och SVD-fallen, medan algoritmerna är rätt olika.

```
% hundhussesvd,  datakomprimering med trunkerad SVD
  clear, clf
  colormap(gray(21)), load Hundhusse, A=Hundhusse;
  [U,S,V]=svd(A); s=diag(S);
  Aapp=zeros(size(A));  nr=0;
  for i=1:15, Aapp=Aapp+s(i)*U(:,i)*V(:,i)';
    if rem(i,5)==0, nr=nr+1;
      subplot(2,3,nr), imagesc(Aapp), axis image
      title([int2str(i) ' singv.']), drawnow
  end, end
```



I medicinsk och teknisk bildbehandling med dator är både fouriertekniken och singulärvärdesfaktoriseringen viktiga verktyg.

(För vidare studier inom bildbehandlingsområdet hänvisas till datorseendekursen som ges av Nada.)

Sakregister