KTH Engineering Sciences

# Approximationsteori

Låt $f$ vara en kontinuerlig funktion som vi vill approximera med en enklare funktion $\tilde{f}(x)$. Vi kommer använda två olika approximationsmetoder: interpolation och minstrakvadratanpassning. Vi börjar med interpolation och det enklaste fallet då $\tilde{f}(x)$ är ett polynom.

## 1 Polynominterpolation

Vid polynominterpolation vill vi hitta ett polynom $p(x)$ som överensstämmer med $f$ i ett antal noder ($x$-koordinater), $x_0 < x_1 < \ldots < x_n$, dvs

$$p(x_j) = f(x_j), \qquad j = 0, \ldots, n.$$

Man kan först fråga sig om detta verkligen alltid är möjligt. Svaret är ja och ges i precis form av Sats 3.2 på sid 141 i Sauer. Vi behöver bara se till att polynomets gradtal är minst lika stort som antalet interpolationsnoder minus ett. I vårt fall är antal noder $n+1$ och vi behöver alltså i allmänhet ett polynom av gradtal $n$. (Bland alla polynom av gradtal mindre eller lika med $n$ som löser problemet är polynomet också unikt; tex, givet två punkter finns det exakt en rak linje som passerar igenom båda två.)

Nästa fråga är hur man kan få fram polynomet numeriskt. För att göra det antar vi att gradtalet är $n$ och ansätter

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n. \tag{1}$$

Vi vill hitta värdena för koefficienterna $c_k$. Villkoren för interpolation blir

$$p(x_j) = \sum_{k=0}^{m} c_k x_j^k = f(x_j), \qquad j = 0, \ldots, n. \tag{2}$$

Detta kan skrivas som ett linjärt ekvationsystem for koefficienterna $c_k$,

$$A\boldsymbol{c} = \boldsymbol{f}, \tag{3}$$

där

$$\boldsymbol{c} = \begin{pmatrix} c_0 \\ \vdots \\ c_n \end{pmatrix}, \qquad \boldsymbol{f} = \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{pmatrix}, \qquad A = \begin{pmatrix} 1 & x_0 & \cdots & x_0^m \\ 1 & x_1 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^n \end{pmatrix}.$$

Notera att $\boldsymbol{c} \in \mathbb{R}^{n+1}$, $\boldsymbol{f} \in \mathbb{R}^{n+1}$ och $A \in \mathbb{R}^{(n+1)\times(n+1)}$. Matrisen $A$ kallas en *Vandermonde*-matris för noderna. Vi kan nu beräkna de okända koefficienterna $c_k$ genom att lösa det linjära ekvationssystemet.

Ett problem med denna metod är att elementen i Vandermondematrisen kan bli väldigt stora om vi har många noder, dvs stort $n$. Det gör matrisen illa konditionerad och metoden blir

opålitligt för stora problem. En bättre variant är att ändra ansatsen (1) till den så kallade Newton-ansatsen,

$$p(x) = d_0 + d_1(x-x_0) + d_2(x-x_0)(x-x_1) + d_3(x-x_0)(x-x_1)(x-x_2) + \cdots + d_n \prod_{j=0}^{n-1}(x-x_j). \quad (4)$$

Denna ansats utnyttjar mer information om problemet när den även inkluderar nodernas värden. Den ger fortfarande ett polynom av gradtal $n$. På samma sätt som ovan leder den till ett linjärt ekvationssystem

$$A_{\text{Newton}}\boldsymbol{d} = \boldsymbol{f}, \quad (5)$$

där

$$\boldsymbol{d} = \begin{pmatrix} d_0 \\ \vdots \\ d_n \end{pmatrix}, \qquad A_{\text{Newton}} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & (x_1-x_0) & 0 & \cdots & 0 \\ 1 & (x_2-x_0) & (x_2-x_0)(x_2-x_1) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (x_n-x_0) & (x_n-x_0)(x_n-x_1) & \cdots & \prod_{j=0}^{n-1}(x_n-x_j) \end{pmatrix}.$$

Denna matris är dels bättre konditionerad än Vandermondematrisen, dels triangulär och det linjära ekvationssystemet är därför billigare att lösa. Ett enkelt sätt att lösa systemet är att använda Newtons dividerade differenser (se Sauer kap 3.1.2), vilket är ekvivalent med vanlig bakåtsubstitution (Gausselimination) för (5).

### 1.1 Interpolationsfel

För att analysera felet vi gör vid polynominterpolation antar vi att alla noder ligger i intervallet $[a,b]$. Man kan då visa att (se Sats 3.4 på sid 152 i Sauer), om $f$ är $n+1$ gånger kontinuerligt deriverbar i $[a,b]$ (dvs $f \in C^{n+1}([a,b])$), och $p(x)$ är det polynom (av max grad $n$) som interpolerar $f$ i $x_0, \ldots, x_n$, ges felet av

$$\boxed{f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)\cdots(x-x_n),} \quad (6)$$

för något $\xi \in [a,b]$, när $x \in [a,b]$. (Detta kan ses som en generalisering av medelvärdessatsen, vilken ges för $n = 0$.) Om noderna är ekvidistant placerade, dvs om

$$x_j = a + jh, \qquad h = \frac{b-a}{n}.$$

kan man också visa (se appendix) att

$$|(x-x_0)\cdots(x-x_n)| \le n!\frac{h^{n+1}}{4}. \quad (7)$$

Definera felet $E_n$ vid $n$ noder som det maximala felet i intervallet,

$$E_n = \max_{a \le x \le b} |f(x) - p(x)|.$$

För $n$ ekvidistanta noder ger då (6) och (7) tillsammas att

$$\boxed{E_n \le \max_{a\le\xi\le b} \frac{|f^{(n+1)}(\xi)|}{4(n+1)} h^{n+1}.} \quad (8)$$

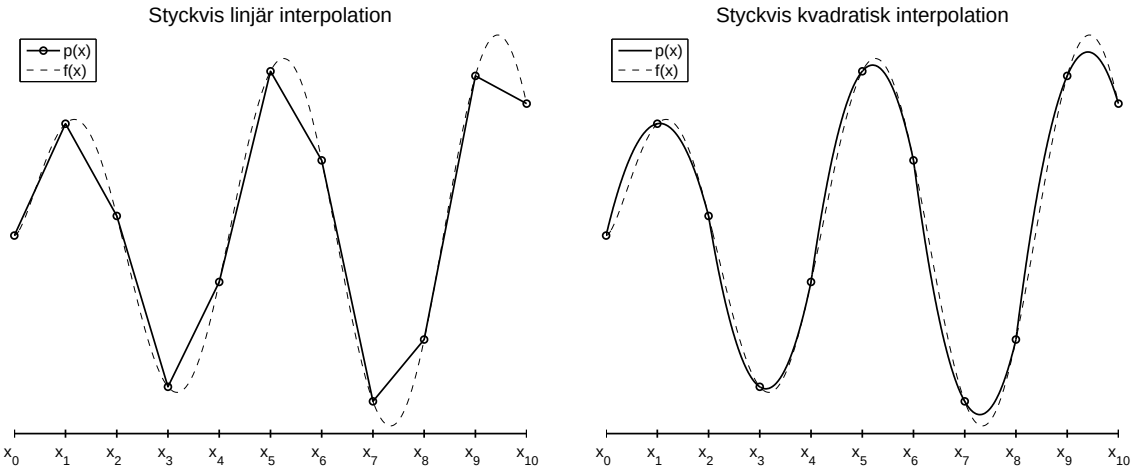


**Figure 1.** Styckvis polynominterpolation: linjär (vänster) och kvadratisk (höger).

då $f \in C^{n+1}([a,b])$ och $p(x)$ av max grad $n$ interpolerar $f$ i noderna. Betyder då detta att felet går mot noll när vi tar många punkter, dvs går $E_n \to 0$ då $n \to \infty$? Nej, i allmänhet inte. Visserligen går $h^{n+1} \to 0$, men ofta går $\max|f^{(n+1)}| \to \infty$ snabbare och $E_n$ kan inte begränsas med (8). För ekvidistant interpolation får interpolationspolynomet vilda oscillationer vid intervallets ändpunkter när $n$ blir stort. Detta kallas *Runges fenomen* och illustreras i Sauer kap 3.2.3.

Slutsatsen är att ekvidistant interpolation med höga gradtal inte är bra. Om man själv kan välja noder är ett bättre alternativ ta noderna icke ekvidistant. Det optimala valet ges av de sk Chebyshev-noderna, se kap 3.3.1 i Sauer. Ett annat mycket praktiskt alternativ är att använda styckvis polynominterpolation med lågt gradtal.

### 1.2 Styckvis polynominterpolation

Styckvis polynominterpolation innebär att vi delar in intervallet $[a,b]$ i många små intervall och interpolerar med lågt gradtal i vart och ett av dessa. Det enklaste fallet är styckvis linjär interpolation där vi använder förstagradspolynom för att interpolera mellan varje nodpar: ett förstgradspolynom interpolerar punkterna $(x_0, f(x_0))$ och $(x_1, f(x_1))$, ett annat $(x_1, f(x_1))$ och $(x_2, f(x_2))$, etc. Det innebär att vi helt enkelt approximerar genom att dra raka streck mellan punkterna, vilket är det typiska sättet en kurva plottas på, tex i MATLAB, se Figur 1, vänster. Samma ide kan förfinas, och istället för förstagradspolynom kan vi tex använda andragradspolynom. Vi kommer då ha ett polynom som interpolerar tre punkter i taget: ett för $(x_0, f(x_0))$, $(x_1, f(x_1))$ och $(x_2, f(x_2))$, ett annat för $(x_2, f(x_2))$, $(x_3, f(x_3))$ och $(x_4, f(x_4))$, etc., se Figur 1, höger. Ytterligare en annan variant av styckvis polynominterpolation är splines som beskrivs i Sauer kap 3.4.

För att göra en uppskattning av felet vid styckvis polynominterpolation kan vi kan använda uppskattningen (8) som vi gjorde tidigare, och applicera den på varje enskilt intervall. För styckvis linjär interpolation använder vi den med $n = 1$ eftersom vi har ett förstagradspolynom

i varje intervall. Det ger

$$E_{\text{lin}} = \max_{a\le x\le b} |f(x) - p_{\text{lin}}(x)| = \max_j \max_{x_j\le x\le x_{j+1}} |f(x) - p_{\text{lin}}(x)| \le \max_j \max_{x_j\le \xi\le x_{j+1}} \frac{|f''(\xi)|}{8} h^2$$
$$= \max_{a\le \xi\le b} \frac{|f''(\xi)|}{8} h^2,$$

om $f \in C^2([a,b])$. Här är alltså $n$ i (8) konstant lika med ett och $h$ går mot noll oberoende av $n$. Därmed konvergerar $E_{\text{lin}} \to 0$ som $O(h^2)$. Noggrannhetsordningen är alltså två för styckvis linjär interpolation. I allmänhet för styckvis interpolation med gradtal $p$ är noggrannhetsordningen $p+1$. För kubiska splines är tex noggrannhetsordningen fyra och felet avtar som $O(h^4)$.

*Slutkommentarer, interpolation:*

- När polynomet är givet på formen (1) kan det evalueras effektivt. Värdet $y = p(x) = c_0 + c_1 x + \cdots c_n x^n$ beräknas med hjälp av *Horners schema* enligt följande

  $y = c_n$
  `for` $k = n-1, n-2, \ldots, 0$
  $\quad y = c_k + xy$
  `end`

  Beräkningskostnaden blir $n$ multiplikationer och $n$ additioner, dvs $O(n)$ flops. Om polynomet är givet på Newton-form (4) kan samma algoritm användas, om $c_n$ byts mot $d_n$ och raden $y = c_k + xy$ byts mot $y = d_k + (x - x_k)y$. Kostnaden blir fortfarande $O(n)$.

- Lagrange-polynomen för noderna är definerade som

  $$L_j(x) := \prod_{k=0, k\ne j}^{n} \frac{x - x_k}{x_j - x_k}, \qquad j = 0, \ldots, n.$$

  De är polynom av gradtal $n$ med egenskapen att

  $$L_j(x_k) = \begin{cases} 1, & j = k, \\ 0, & j \ne k. \end{cases}$$

  Med dessa kan interpolationsproblemet enkelt lösas,

  $$p(x) = \sum_{j=0}^{n} L_j(x) f(x_j).$$

  Denna form för interpolationspolynomet är dock krånglig att använda i numeriska beräkningar. Evaluering av polynomet och dess derivata, samt beräkning av dess integral har en hög berärkningskostnad jämfört med formerna (1) och (4). Lagrange-polynomen är snarare ett teoretiskt verktyg och har ett stort värde vid analys av approximationsproblem.

- Några praktiska MATLAB-funktioner för interpolation är `c=polyfit(x,f,n)`, som ger $c_k$-koefficienterna när $x_k$ och $f(x_k)$ ligger i $(n+1)$-vektorerna `x` och `f`; `polyval(c,x)` som evaluerar ett polynom med koefficienterna `c` i punkten `x`; samt `interp1(xnod,fnod,x,typ)` som evaluerar styckvis polynominterpolation av `(xnod,fnod)` i punkten `x`, med varierande interpolationstyp som bestäms av sista argumentet. Tex kan `typ` vara `'linear'`, `'cubic'`, eller `'spline'`. Se MATLABs `help`-kommando för mer info.

## 2 Minstakvadratanpassning

Vid minstakvadratanpssning söker vi en enkel funktion som approximerar $f(x)$ genom att minimera den totala avvikelsen från $f$ i ett antal givna noder. Mer precist söker vi en funktion $\tilde{f}$ som minimerar "felkvadratsumman",

$$r = \sum_{j=0}^{n} (\tilde{f}(x_j) - f(x_j))^2.$$

Vi börjar med att anta att $\tilde{f}$ är ett polynom. Om $f$ tex är en snäll funktion med brus eller andra störningar överlagrade kommer vanlig polynominterpolation, både styckvis och direkt, ge en mycket oscillerande approximation. Minstakvadratlösningen med ett polynom av lågt gradtal ger en snällare lösning.

För att beräkna minstakvadratlösningen för ett polynom gör vi som tidigare ansatsen

$$p(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_m x^m,$$

där $m$ nu är mindre än $n$. Vi kan då inte längre hitta $c_k$ så att $p(x_j) = f(x_j)$ för alla $j$. Istället vill vi minimera skillnaden. Liksom ovan kan vi skriva denna på matrisform,

$$A\boldsymbol{c} - \boldsymbol{f} = \begin{pmatrix} p(x_0) - f(x_0) \\ \vdots \\ p(x_n) - f(x_n) \end{pmatrix}, \qquad \overbrace{\square}^{m+1} \; \square - \square \Big\} n+1. \tag{9}$$

Matrisen blir alltså nu rektangulär, $A \in \mathbb{R}^{(n+1)\times(m+1)}$, $\boldsymbol{c} \in \mathbb{R}^{m+1}$ och $\boldsymbol{f} \in \mathbb{R}^{n+1}$. Systemet $A\boldsymbol{c} = \boldsymbol{f}$ är överbestämt och saknar då i allmänhet lösningar. Vi noterar dock att eftersom

$$||A\boldsymbol{c} - \boldsymbol{f}||_2^2 = \sum_{j=0}^{n} (p(x_j) - f(x_j))^2 = r,$$

är minimering av felkvadratsumman $r$ ekvivalent med att lösa $A\boldsymbol{c} = \boldsymbol{f}$ i minstakvadratmening, dvs hitta den vektor $\boldsymbol{c}$ som minimerar den euklidiska normen av residualen,

$$||A\boldsymbol{c} - \boldsymbol{f}||_2 = \min_{\boldsymbol{b}\in\mathbb{R}^{m+1}} ||A\boldsymbol{b} - \boldsymbol{f}||_2. \tag{10}$$

Vi ska nu visa att lösningen $\boldsymbol{c}$ till minstakvadratproblemet ges av *normalekvationerna*

$$\boxed{A^T A\boldsymbol{c} = A^T \boldsymbol{f}}. \tag{11}$$

Notera att $A^T A \in \mathbb{R}^{(m+1)\times(m+1)}$ är en kvadratisk matris och systemet har alltid en unik lösning när kolumnerna i $A$ är linjärt oberoende (se nedan).

Normalekvationerna kan härledas på följande sätt. Antag att $\boldsymbol{c}$ löser (11). Eftersom $||\boldsymbol{x}||_2^2 = \boldsymbol{x}^T\boldsymbol{x}$ har vi då för alla $\boldsymbol{b} \in \mathbb{R}^{m+1}$,

$$\begin{aligned} ||A\boldsymbol{b} - \boldsymbol{f}||_2^2 - ||A\boldsymbol{c} - \boldsymbol{f}||_2^2 &= \boldsymbol{b}^T A^T A\boldsymbol{b} - 2\boldsymbol{b}^T A^T \boldsymbol{f} + \boldsymbol{f}^T\boldsymbol{f} - \boldsymbol{c}^T A^T A\boldsymbol{c} + 2\boldsymbol{c}^T A^T \boldsymbol{f} - \boldsymbol{f}^T\boldsymbol{f} \\ &= \boldsymbol{b}^T A^T A\boldsymbol{b} - 2\boldsymbol{b}^T A^T \boldsymbol{f} - \boldsymbol{c}^T A^T A\boldsymbol{c} + 2\boldsymbol{c}^T A^T \boldsymbol{f} = \quad \{\text{utnyttja (11)}\} \\ &= \boldsymbol{b}^T A^T A\boldsymbol{b} - 2\boldsymbol{b}^T A^T A\boldsymbol{c} + \boldsymbol{c}^T A^T A\boldsymbol{c} \\ &= ||A(\boldsymbol{b} - \boldsymbol{c})||_2^2 \geq 0. \end{aligned}$$

Med andra ord är

$$||A\boldsymbol{b} - \boldsymbol{f}||_2^2 \geq ||A\boldsymbol{c} - \boldsymbol{f}||_2^2 \qquad \forall \boldsymbol{b} \in \mathbb{R}^{m+1},$$

vilket visar att $\boldsymbol{c}$ faktiskt minimerar residualen och uppfyller (10).

Alternativt kan vi härleda (11) med vanlig flervariabelanalys. Definiera

$$\Phi(\boldsymbol{b}) := ||A\boldsymbol{b} - \boldsymbol{f}||_2^2.$$

Låt $\nabla$ betyda gradienten med avseende på $\boldsymbol{b}$. Då uppfyller minimumet $\boldsymbol{c}$

$$\nabla\Phi(\boldsymbol{c}) = 0.$$

Vi utnyttjar att

$$\nabla(\boldsymbol{b}^T U\boldsymbol{b}) = 2U\boldsymbol{b}, \qquad \nabla \boldsymbol{u}^T\boldsymbol{b} = \boldsymbol{u},$$

för alla kvadratiska matriser $U \in \mathbb{R}^{(m+1)\times(m+1)}$ och vektorer $\boldsymbol{u} \in \mathbb{R}^{m+1}$. (Jämför skalära fallet där $(ub^2)' = 2ub$ och $(ub)' = u$.) Detta ger

$$\nabla\Phi(\boldsymbol{b}) = \nabla\left(\boldsymbol{b}^T A^T A\boldsymbol{b} - 2\boldsymbol{b}^T A^T\boldsymbol{f} + \boldsymbol{f}^T\boldsymbol{f}\right) = 2A^T A\boldsymbol{b} - 2A^T\boldsymbol{f}.$$

Följdaktligen har vi

$$0 = \nabla\Phi(\boldsymbol{c}) = 2A^T A\boldsymbol{c} - 2A^T\boldsymbol{f},$$

vilket är samma sak som (11). Det återstår att visa att $\boldsymbol{c}$ är ett minimum (och inte ett maximum eller en sadelpunkt, till exempel). Villkoret för detta är att Hessianen till $\Phi$ är en positivt definit matris vid $\boldsymbol{c}$. I vårt fall är Hessianen precis $2A^T A$ och vi behöver alltså visa att

$$\boldsymbol{x}^T A^T A\boldsymbol{x} > 0, \qquad \text{för alla } \boldsymbol{x} \neq 0 \text{ i } \mathbb{R}^{m+1}. \tag{12}$$

Vi noterar nu att vårt tidigare antagande om att kolumnerna i $A$ är linjärt oberoende betyder att $A\boldsymbol{x} \neq 0$ när $\boldsymbol{x} \neq 0$. Därmed följer (12) från det faktum att $\boldsymbol{x}^T A^T A\boldsymbol{x} = ||A\boldsymbol{x}||_2^2$. Att $A^T A$ är positivt definit implicerar också att $A^T A$ är ickesingulär och att normalekvationerna (11) har en unik lösning.

Minstakvadratproblem i MATLAB:

- `c=polyfit(x,f,m)` ger minstakvadratlösningen om $m < n$ och vektorerna `x` och `f` har längd $n + 1$.
- `A\b` ger minstakvadratlösningen om `A` är en $n \times m$-matris, `b` en $n$-vektor och $m < n$.

Se MATLABs `help`-kommando för mer info.

## 3 Interpolation med andra basfunktioner

Vi betraktar ett allmännare fall när $\tilde{f}(x)$ är en linjärkombination av godtyckliga basfunktioner, dvs när den kan skrivas som

$$\tilde{f}(x) = \sum_{k=0}^{m} c_k\psi_k(x),$$

med några funktioner $\psi_k(x)$ och obekanta koefficienter $c_k \in \mathbb{R}$, $k = 0, \ldots, m$. I polynomfallet använder vi till exempel $\psi_k(x) = x^k$. I allmänhet kan man använda välja andra funktioner som liknar $f$, tex trigonometriska funktioner $\psi_k(x) = \exp(ikx)$ om $f$ är periodisk. Interpolationsvillkoren blir då

$$\tilde{f}(x_j) = \sum_{k=0}^{m} c_k\psi_k(x_j) = f(x_j), \qquad j = 0, \ldots, n. \tag{13}$$

Liksom tidigare kan det skrivas som ett linjärt ekvationsystem for $c_k$,

$$A\boldsymbol{c} = \boldsymbol{f}, \tag{14}$$

där

$$\boldsymbol{c} = \begin{pmatrix} c_0 \\ \vdots \\ c_m \end{pmatrix}, \qquad \boldsymbol{f} = \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{pmatrix}, \qquad A = \begin{pmatrix} \psi_0(x_0) & \psi_1(x_0) & \cdots & \psi_m(x_0) \\ \psi_0(x_1) & \psi_1(x_1) & \cdots & \psi_m(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_0(x_n) & \psi_1(x_n) & \cdots & \psi_m(x_n) \end{pmatrix}.$$

Vi antar att basfunktionerna $\psi_k(x)$ och noderna $\{x_i\}$ är valda så att kolumnerna i $A$ är linjärt oberoende. När $n = m$ är då $A$ ickesingulär och vi kan hitta $\boldsymbol{c}$ som uppfyller (14) och alltså $c_k$ som uppfyller (13). När antalet punkter är större än antalet obekanta, dvs då $n > m$, kan vi lösa systemet i minstakvadratmening som ovan, vilket ger det $\tilde{f}$ som minimerar

$$\sum_{j=0}^{n} (\tilde{f}(x_j) - f(x_j))^2.$$

## A Bevis av (7)

För $x \in (a, b)$ kan vi skriva $x = a + (j_* + \alpha)h$, med $0 \le \alpha < 1$ och $j_* \in \{0, \ldots, n-1\}$. Eftersom vi har ekvidistant placerade noder får vi

$$\begin{aligned} |(x - x_0) \cdots (x - x_n)| &= |(j_*h + \alpha h)(j_*h + \alpha h - h) \cdots (j_*h + \alpha h - nh)| \\ &= h^{n+1} |(j_* + \alpha)(j_* - 1 + \alpha) \cdots (j_* - n + \alpha)| \\ &= h^{n+1} |(j_* + \alpha) \cdots (1 + \alpha)\alpha(1 - \alpha)(2 - \alpha) \cdots (n - j_* - \alpha)| \\ &\le h^{n+1}(j_* + 1)!(n - j_*)!\alpha(1 - \alpha) = h^{n+1}(j_* + 1)!(n - j_*)! \left( \frac{1}{4} - \left( \alpha - \frac{1}{2} \right)^2 \right) \\ &\le \frac{h^{n+1}}{4}(j_* + 1)!(n - j_*)! \le \frac{h^{n+1}}{4} n!. \end{aligned}$$

Det sista steget är uppenbart då $j_* = 0$ eller $j_* = n - 1$. För övriga fall följer det av att

$$\frac{(a+1)!b!}{(a+b)!} = \frac{1 \cdot 2 \cdot \cdots \cdot (a+1)}{(b+1) \cdots (a+b)} = \frac{1 \cdot 2 \cdot \cdots \cdot \min(a+1, b)}{\max(b+1, a+2) \cdots (a+b)} < 1, \qquad \text{när } a > 0 \text{ och } b > 1.$$