AlMa II - Überblick Numerik

Jochen Garcke

universitätbonn

Fraunhofer SCAI

Polynominterpolation

- für Funktion f finde Polynom $p \in \Pi_n$ mit $p(x_j) = f(x_j), j = 0, \ldots, n$
- Existenz, Eindeutigkeit
  z.B. per Vandermonde-Matrix, Dimension des Vektorraums + l.u.
- verschiedene Basen für $p(x) = \sum \alpha_i \phi_i(x)$:
  - Monom-Basis x^i (z.B. für Vandermonde-Matrix, Beweise)
  - Lagrange-Basis mit $L_i(x_j) = \delta_{ij}$ (damit $\alpha_i = f(x_i)$)
  - Newton-Basis $\omega_i(x) = \pi_{j=0}^{i-1}(x - x_j)$ (Newton-Interpolationsverfahren)
- Satz von Aitken liefert Rekursionsformel
- verschiedene Algorithmen:
  - "könnten" LGS mit Vandermonde-Matrix lösen
  - "könnten" L_i explizit berechnen, kennen α_i
  - Schema von Neville als rekursive Formel zur Auswertung weniger $p(x)$
  - Newton-Interpolationsverfahren zur Bestimmung der α_is
- Newton-Interpolation nutzt dividierte Differenzen, Bezug zu Aitken
- Hermite-Interpolation kann auch Auswertung der Ableitung nutzen
  - basiert auf verallgemeinerten dividierten Differenzen
  - Analyse und Algorithmen laufen damit dann analog

Algorithmen zur Polynominterpolation I

Algorithm 1: Neville-Schema

Data: n, Knoten x(), Auswertungen fx() (Vektoren der Länge n+1), t
Result: Auswertung an der Stelle t des Interpolationspolynoms p def. durch x,fx

```
for i=0,...,i <= n do
    p(i) = fx(i)        ▷ Initialisierung / erste Spalte, p=fx in kurz
for k=1,...,k <= n do
    for i=0,...,i <= n-k do
        p(i) = ((t - x(i))*p(i+1) - (t - x(i+k)) * p(i)) / (x(i+k) - x(i))
return p(0)
```

Algorithmen zur Polynominterpolation II

Algorithm 2: Newton-Interpolation - Berechnung der Koeffizienten

Data: n, Knoten x(), Auswertungen fx() (Vektoren der Länge n+1)
Result: Berechnung der Koeffizienten a() bzgl. der Newton-Basis des Interpolationspolynoms p(t) def. durch x,fx

```
for i=0,...,i <= n do
    a(i) = fx(i)      ▷ Initialisierung / erste Spalte, a=fx in kurz
for k=1,...,k <= n do
    for i=n,n-1,...,i >= k do
        a(i) = (a(i) - a(i-1)) / (x(i) - x(i-k))
return a()
```

Innere Schleife kann auch Vorwärtslaufen wie beim Neville-Schema, dann aber mit zusätzlichem Vektor zum Speichern der Koeffizienten

Algorithmen zur Polynominterpolation III

Algorithm 3: Horner-Schema zum Auswerten der Newton-Interpolierenden

Data: Knoten x(), Koeffizienten a() (Vektoren der Länge n+1), t
Result: Auswertung an der Stelle t des Newton-Interpolationspolynoms
p def. durch x, a

```
p = a(n)
for k=n-1,n-2,...,k >= 0 do
    p = a(k) + (t - x(k))*p
```

Approximationsfehler

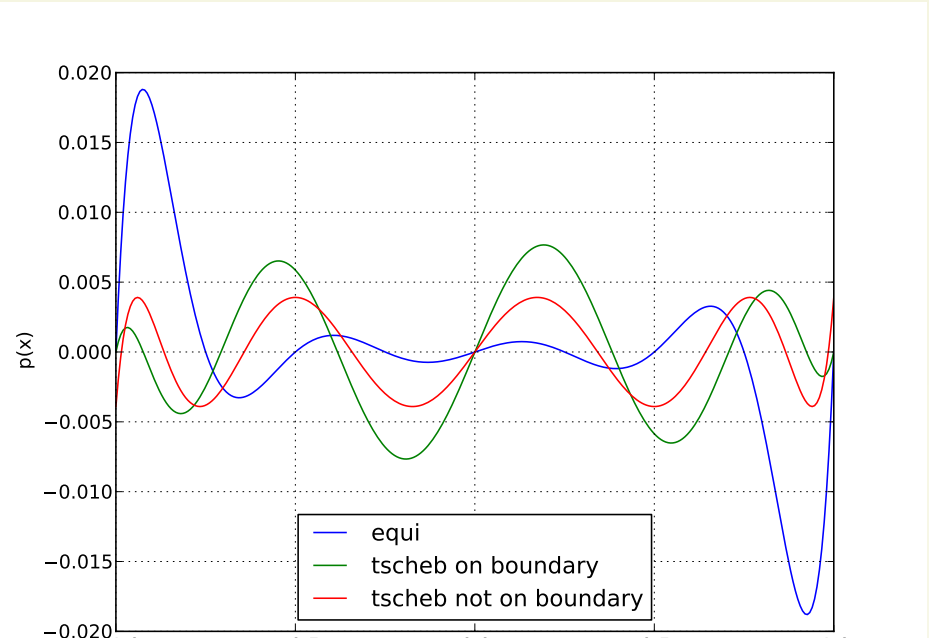
Approximationsfehler der Polynominterpolation

Für $f \in C^{n+1}([a,b])$, $x_i \in [a,b], 0 \leq i \leq n$ und $x \in [a,b]$ gilt

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\tau)}{(n+1)!}\omega_{n+1}(x)$$

für ein $\tau \in [a,b]$

Abhängigkeit von $\omega(x)$ von Knotenwahl: $w_n(x)$ für Satz 8.10 mit $n = 8$



Kondition

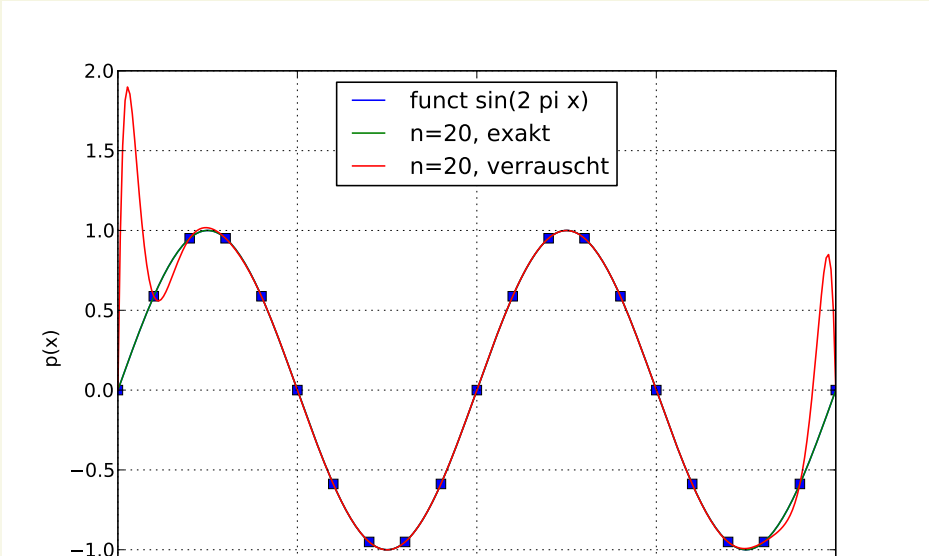
Konditionszahl der Polynominterpolation

Für $x_i \in [a, b], 0 \leq i \leq n, x_i < x_{i-1}$ ist bzgl. Supremumsnorm die Kondition der Polynominterpolation

$$\kappa_{abs} = \Lambda_n := \max_{x \in [a,b]} \sum_{i=0}^{n} |L_i(x)|$$

mit L_i den zugehörigen Lagrange-Polynomen.

Auswirkung der Kondition auf Datenfehler



Stückweise lineare Interpolation

- alternative zu höheren Polynomgraden sind äquidistante Knoten, auf denen dann stückweise (linear) interpoliert wird
- Approximationsausgaben sind dann in der L^2-Norm

$$\|f\|_{L^2}^2 := \int_a^b |f(x)|^2 dx = \langle f, f \rangle_{L^2}$$

- für $f \in C^2$ mit stückweise linear Interpolierenden s gilt (mit $h = (b-a)/n$)

$$\|f - s\|_{L^2} \leq \frac{h^2}{2} \|f''\|_{L^2}$$

bzw.

$$\|(f - s)'\|_{L^2} \leq \frac{h}{\sqrt{2}} \|f''\|_{L^2}$$

Numerische Integration / Quadratur

- wir kennen Quadraturformeln mit m, w_j, x_j fest gewählt

$$Q[f] = \sum_{j=0}^{m} w_j f(x_j)$$

  Die Formel ist ein Algorithmus.
- insbesondere Newton-Cotes-Formeln mit

$$w_i := \int_a^b L_i(x)\,dx$$

  auf Basis von äquidistanten Knoten $x_j = a + \frac{b-a}{m} j, \quad j = 0, \ldots, m$
- aus $Q[\cdot]$ zusammengesetzte Quadraturformel $Q_n[f]$, bei der Q auf n gleich große Teilintervalle von $[a, b]$ angewandt wird (auch das ist ein Algorithmus)
- Exaktheitsgrad q über Polynome: $Q[p] = I[p] \quad \forall p \in \Pi_q$
- Konvergenz der Ordnung s: $|Q_n[f] - I[f]| = \mathcal{O}(n^{-s})$

Trapez- und Simpson-Regel

- Trapezformel
  für $m = 1$ Gewichte(/h) sind 1/2 und 1/2, mit Fehler $\frac{1}{12} h^3 \| f'' \|_\infty$
- Simpson-Regel
  für $m = 2$ Gewichte(/h) sind 1/6, 4/6 und 1/6, mit Fehler $\frac{1}{90} h^5 \| f^{(4)} \|_\infty$
- entsprechende zusammengesetzte Formeln, mit entsprechenden Fehlerabschätzungen
- wobei Simpson-Regel gerne mit $h = (b - a)/2n$ gemacht wird
- Beweise haben was mit stückweiser Polynominterpolation zu tun

Richardson-Extrapolation

- Richardson-Extrapolation basiert auf asymptotischer Entwicklung

$$\mathcal{A}(h) = \alpha_0 + \alpha_1 h^{p_1} + \alpha_2 h^{p_2} + \ldots + \alpha_K h^{p_K} + \mathcal{O}(h^{p_{K+1}})$$

  mit α_i unabhängig von h und $0 < p_1 < p_2 < \ldots < p_{K+1}$
- mit $0 < \delta < 1$ gilt dann im ersten Schritt

$$\alpha_0 = \frac{\delta^{-p_1}\mathcal{A}(h) - \mathcal{A}(\delta^{-1}h)}{\delta^{-p_1} - 1} + \mathcal{O}(h^{p_2}) = \mathcal{A}(h) + \frac{\mathcal{A}(h) - \mathcal{A}(\delta^{-1}h)}{\delta^{-p_1} - 1} + \mathcal{O}(h^{p_2})$$

  d.h. für $\delta^{-p_1}\mathcal{A}(h), \mathcal{A}(\delta^{-1}h)$ löschen sich $\delta^{-p_1}\alpha_1 h^{p_1}$ und $\alpha_1\delta^{-p_1}h^{p_1}$ aus
- das Auslöschen der "ersten" Fehlerordnung wird wiederholt
- mit der Euler-Maclaurinschen Summenformel wissen wir von einer asymptotischen Entwicklung (mit $p_i = 2 * i$) für die Trapezformel
- Wahl von $\delta = 1/2$, $h = b - a$ ergibt die Romberg-Integration

Algorithmus der Romberg-Integration

Algorithm 4: Romberg-Integration

Data: Intervallgrenzen a,b, Schritte m, Funktionsauswertungen f() an $2^m + 1$ äquidistanten Punkten

Result: Romberg-Integration von f der Stufe m

for *i=0,...,i<=m* **do**
    h = $(b - a)/2^i$
    schritt = 2^{m-i} ▷ `passender Index-Abstand für diese Trapezformel`
    A(i) = f(0)/2
    for *j=1,...,j<=*$2^i - 1$ **do**
        A(i) = A(i) + f(j*schritt)
    A(i) = h*(A(i) + f(2^m)/2)

for *j=1,...,j<=m* **do**
    faktor = 4^j
    for *i=m,...,i>=j* **do**
        A(i) = A(i) + (A(i) - A(i-1))/(faktor - 1)

return A(m)

Adaptive Simpson-Romberg-Integration

Algorithm 5: Adaptive Simpson-Romberg-Integration (ASRI)

Data: Intervallgrenzen a,b, Funktion f, fx(0)=f(a), fx(2) = f((a+b)/2), fx(4)=f(b), eps

Result: Adaptive Simpson-Romberg-Integration von f mit Abbruchkriterium eps

```
h = (b-a)/2; m = (a+b)/2
S1 = (fx(0) + 4*fx(2) + fx(4))*h/3                    ▷ Simpson-Regel
fx(1) = f((a+h)/2); fx(3) = f((b-h)/2)
S2 = (fx(0) + 4*fx(1) + 2*fx(2) + 4*fx(3) + fx(4))*h/6 ▷ zus.ges.  Simpson
A22 = (16*S2 - S1)/15      ▷ nach Romberg-Integration für Simpson
delta = abs(S2 - A22)                          ▷ Fehlerschätzer
if delta < eps then
    I = A22
else
    I = Alg.5 ASRI(a,m,f,fx(0),fx(1),fx(2),eps)
    I = I + Alg.5 ASRI(m,b,f,fx(2),fx(3),fx(4),eps)
return I
```

Fourier trigonometrische Interpolation

- betrachten periodisches komplexes Interpolationsproblem

$$p(x_k) = y_k, k = 0, \ldots, n-1$$

und nutzen komplexes trigonometrisches Polynom vom Grad $n-1$

$$p(x) = \sum_{j=0}^{n-1} c_j e^{ijx}$$

- wir nutzen äquidistante Stützstellen $x_k = 2\pi k/n$ und haben dann

$$c_j = \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{-jk} y_k, \quad j = 0, \ldots, n-1$$

mit $\omega_n := e^{2\pi i/n}$ n-te komplexe Einheitswurzel

- entsprechend ergeben sich reellwertig Formeln (hier $n = 2m+1$)

$$\frac{a_0}{2} + \sum_{j=1}^{m} (a_j \cos(jx) + b_j \sin(jx)), \quad a_j = 2Re(c_j), b_j = -2Im(c_j)$$

und Summenformel für $a_j = 2/n \sum_{k=0}^{n-1} y_k \cos(jx_k)$, analog b_j mit sin

Diskrete Fourier-Transformation

- diskrete Fourier-Tranformation: $F_n : \mathbb{C}^n \to \mathbb{C}^n$ definiert durch

$$F_n f = g, \quad g_j := \sum_{k=0}^{n-1} e^{-ijx_k} f_k, \quad 0 \leq j \leq n$$

- die DFT kann als Multiplikation eines Vektors f mit der Matrix

$$(F_n)_{kl} := e^{-ikx_l} = \omega_n^{-kl}$$

- basierend auf der Beobachtung $(\omega_n)^2 = \omega_{n/2}$ und geschickter Umordnung gibt es die schnelle Fourier-Transformation (FFT) mit $\mathcal{O}(nlogn)$ Aufwand für die Matrix-Vektormultiplikation $F_n f$

$$F_n = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2} & 0 \\ 0 & F_{n/2} \end{bmatrix} \Pi_n$$

  mit I_n Einheitsmatrix, $D_n = diag(1, e^{-2\pi i/n}, e^{-2\pi i 2/n}, \ldots, e^{-2\pi i(n-1)/n})$, und Permutationsmatrix $\Pi_n v = (v_0, v_2, v_4, \ldots, v_{n-2}, v_1, v_3, \ldots, v_{n-1})^T$
- Algorithmus auch auf Real- bzw Imaginärteil von f anwendbar

FFT Algorithmus

Algorithm 6: FFT

Data: n, f(), s
Result: DFT von f der Größe n
if *n=1* **then**
    g(0) = f(0)
else
    g(0:n/2-1) = FFT(n/2, f(step 2), 2s) ▷ `DFT von f[0], f[2s], ...`
    g(n/2:n-1) = FFT(n/2, f(1 + step 2), 2s) ▷ `DFT von f[1], f[3s], ...`
    for *k=0, ..., k < n/2* **do**
        z = g(k)
        w = exp($\pm 2\pi ik/n$)
        x = w * g(k+n/2)
        g(k) = z + x
        g(k+n/2) = z - x
return g

mit Bitmanipulationen lässt sich die FFT im Platz und ohne Rekursion effizient realisieren

Iterationsverfahren für LGS

- wir suchen x mit $\|x - x^*\| \leq \epsilon \|x^*\|$ für $x^* = A^{-1}b$
- nutzen Iterationsverfahren/Fixpunktgleichung $x = Mx + Nb =: G(x)$

$$x^{(k+1)} = (I - W^{-1}A)x^{(k)} + W^{-1}b$$

  mit $M = W^{-1}(W - A)$, basierend auf Zerlegung $A = W - E$
- mit $A = D - L - R$ ergeben sich per Wahl von
  - $W = D$ Jacobi-Verfahren / Gesamtschrittverfahren
  - $W = D - L$ Gauß-Seidel-Verfahren / Einzelschrittverfahren
  - $W = \tau^{-1}I, \tau > 0$ Richardson-Verfahren
  - $W = \omega^{-1}D - L, \omega \in (0, 2)$ SOR-Verfahren

SOR-Verfahren

Algorithm 7: SOR-Verfahren, omega=1 gibt Gauß-Seidel-Verfahren

Data: A(,), b(), x(), kmax, eps, omega
Result: Approximation für Ax=b mit Residuum eps

```
k = 0; r() = b() - A(,) x(); r0 = norm(r); err = r0; xold = x0
while err > tol und k < kmax do
    k = k +1
    for i = 1, ..., i <= n do
        s = 0
        for j=1,...,j <= i-1 do
            s = s + A(i,j) * x(j)
        for j=i+1,...,j <= n do
            s = s + A(i,j) * xold(j)
        x(i) = omega * (b(i) -s)/A(i,i) + (1 - omega) * xold(i)
    xold() = x(); r() = b() - A(,)x(); err = norm(r)/r0
return x()
```

hier xold nur zur Illustration, programmiert wird mit nur einem Vektor !
Jacobi-Verfahren hingegen benötigt zwei Vektoren xold und x

Konvergenzanalyse

- Konvergenzanalyse basiert auf dem Banachschen Fixpunktsatz
- mit $Mx + Nb$ als Iteration ist das Ziel $\|M\| < 1$ in einer Matrixnorm
- dann q-lineare Konvergenz, sowie a-priori Schranken

$$\|x^{(k)} - x^*\| \leq \frac{q^k}{1-q}\|x^{(1)} - x^{(0)}\|$$

und a-posteriori Schranken

$$\|x^{(k+1)} - x^*\| \leq \frac{q}{1-q}\|x^{(k+1)} - x^{(k)}\|$$

- für diagonaldominante Matrizen A konvergieren Jacobi und Gauß-Seidel
- SOR braucht für Konvergenz $\omega \in (0, 2)$
- für symm. pos.def. konvergiert SOR für $\omega \in (0, 2)$

Iterationsverfahren für nichtlineare Gleichungen

- Verfahren zur Bestimmung einer Nullstelle x^* von f, also $f(x^*) = 0$
- Bisektionsverfahren, Halbierung des Intervalls $[l, r]$ und Wahl der Seite mit Vorzeichenwechsel, funktioniert sofern $f(l)f(r) < 0$
- Regula falsi, neue Intervallgrenze ist Nullstelle der linearen Interpolanten zwischen linker $(l, f(l))$ und rechter Grenze $(r, f(r))$

$$m = l - \frac{r - l}{f(r) - f(l)} f(l) = \frac{lf(r) - rf(l)}{f(r) - f(l)}$$

- beide haben globale, lineare Konvergenz
- Sekantenverfahren als Iterationsverfahren mit beiden letzten Iterierten und linearer Interpolanten der dortigen Funktionswerte

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})} f(x^{(k)})$$

- lokale, aber superlineare Konvergenz, also mit Ordnung $p > 1$

Bisektionsverfahren

Algorithm 8: Bisektionsverfahren

Data: f (stetig auf (a,b), mit f(a)f(b) < 0), eps
Result: x mit $|x - x^*| \leq eps/2$, wobei $f(x^*) = 0$

```
m = (a+b)/2; l=a; r=b
while  l l-r l  > eps do
    if f(m) = 0 then
        STOP
    if  f(l)f(m) < 0 then
        r = m
    else
        l = m
    m = (l+r)/2
return m
```

für Regula falsi ersetze im Algorithmus (und hat $|x - x^*| \leq eps$)

$$m = l - \frac{r - l}{f(r) - f(l)} f(l) = \frac{lf(r) - rf(l)}{f(r) - f(l)}$$

Sekantenverfahren

Algorithm 9: Sekantenverfahren

Data: f (stetig auf (a,b)), x0, x1, eps, nmax
Result: x als Näherung von x^*

```
err = eps +1
fx1 = f(x1); fx0 = f(x0)
while n < nmax und err > eps do
    n = n +1
    x = x1 - fx1 (x1 - x0)/(fx1 - fx0)
    x0 = x1
    xd0 = fx1
    x1 = x
    fx1 = f(x1)
    err = abs(x0 - x1)
return m
```

Newton-Verfahren

- basiert auf Linearisierung der nichtlinearen Funktion f mit $f(x) = 0$ mittels Taylor

$$0 = f(x^*) \approx f(x^{(0)}) + f'(x^{(0)})(x^* - x^{(0)})$$

- ergibt als Iterationsverfahren

$$x^{(k+1)} = x^{(k)} - (f'(x^{(k)}))^{-1} f(x^{(k)})$$

- sofern $\|(f'(z))^{-1}\|_M \leq \alpha$ und f' Lipschitzstetig mit Konstante β gilt in einer Kugel um x^* mit Radius $2/(\alpha\beta)$ quadratische Konvergenz (sofern sich alles in einem konvexen offenem Gebiet befindet)

$$\|x^{(k+1)} - x^*\|_V \leq \frac{\alpha\beta}{2} \|x^{(k)} - x^*\|_V^2$$

- Newton hat somit schnelle Konvergenz, aber nur lokal und mit größerem Berechnungsaufwand

Algorithmus Newton-Verfahren

Algorithm 10: Newton-Verfahren zur approximativen Lösung nichtlinearer Gleichungen

Data: f (stetig diff'bar), x0, eps, kmax
Result: x mit $|x^{(k+1)} - x^{(k)}| \leq eps$ oder $k \geq kmax$
err = eps +1
x=x0
while *k < kmax und err > eps* **do**
    W = f'(x(k))
    löse Wp = - f(x) ▷ `z.B. LR-Zerlegung, oder iteratives Verfahren`
    x = x + p
    k = k+1
    err = norm(p)
return x