

Zusammenfassung vom 17.04.2024

Nullstellensuche

- Bisektionsverfahren - Teilen eines Startintervalls, in dem Vorzeichen von f wechselt
- Newtonverfahren - lineare Näherung an die Funktion (benötigt Ableitung)
- Sekantenverfahren - lineare Näherung an die Funktion (numerische Ableitung)

Bisektionsverfahren erfordert viele Schritte

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Newton/Sekantenverfahren konvergieren nicht/schlecht in der Nähe von Extrema

Numerische Integration: Romberg Verfahren

basiert auf Trapezregel

Integrationsfehler wird zu $h \rightarrow 0$ **extrapoliert**

höhere Genauigkeit bei vergleichsweise wenigen Stützstellen

Möglichkeit **adaptiven Code** zu schreiben

Extrapolation wurde mit Hilfe eines Polynoms durchgeführt:

Neville Schema zur Bestimmung des Polynoms

ermöglicht bessere Skalierungen mit h ohne Gewichte mit unterschiedlichem Vorzeichen zu benutzen

Gauß Integration

Ein Nachteil des Romberg Verfahrens ist, dass es keinen Satz Stützpunkte und Gewichte gibt.

Die **Gauß Integration** ist eine Methode, die zu **Stützstellen und Gewichten** führt. Für glatte Funktionen erhält man hier eine extrem gute Näherung an das Integral.

Bisher einfache Verfahren (Trapez, Simpson, ...):

- äquidistante Stützstellen
- sehr hohe Ordnungen sind nicht brauchbar, da Gewichte negativ werden

Jetzt zeigen wir, dass durch nicht äquidistante Stützstellen auch bei hohen Ordnungen rein positive Gewichte gefunden werden.

Problemstellung: Finde optimale x_i und ω_i , so dass:

$$\int_a^b dx \ w(x)f(x) \approx \sum_{i=0}^{n-1} \omega_i \ f(x_i) \quad \text{für } w(x) \geq 0$$

Wir sehen gleich: falls f ein Polynom vom Höchstgrad $2n - 1$ ist : mit x_i als Nullstellen eines Polynoms vom Grad N ist die Approximation exakt!

Dazu ist es hilfreich **orthogonale Polynome** zu betrachten.

Definieren zunächst einen Satz orthogonaler Polynome auf Basis des Skalarproduktes:

$$(f, g) \equiv \int_a^b dx w(x) f(x)g(x)$$

Betrachten wir den Vektorraum Π_{n-1} der Polynome vom Grad $\leq n-1$:

Durch **Gram-Schmidt Orthogonalisierung** kann man eine orthogonale Basis von Π_{n-1} durch die Rekursionsrelation

$$p_{i+1}(x) = (x - a_i) p_i(x) - b_i p_{i-1}(x) \quad \text{mit} \quad a_i = \frac{(xp_i, p_i)}{(p_i, p_i)} \quad b_i = \frac{(p_i, p_i)}{(p_{i-1}, p_{i-1})}$$

erhalten, wobei man mit $p_{-1} = 0$, $p_0 = 1$ starten kann.



der Vektorraum Π_{n-1} ist orthogonal zu p_n ,

da man $q(x) \in \Pi_{n-1}$ durch p_0, \dots, p_{n-1} darstellen kann und so $(q, p_n) = 0$.

Das hat aber die wichtige Konsequenz, dass $p_n(x)$ **n einfache Nullstellen** in $]a,b[$ hat.

Beweisidee

Konstruiere $q(x) = \prod_{j=0}^{k-1} (x - \xi_j)$ wobei ξ_0, \dots, ξ_{k-1} die Nullstellen **ungerader** Vielfachheit von $p_n(x)$ sind, und k die Anzahl solcher Nullstellen ist.

Für ein Polynom $p_n(x)$ $\rightarrow k < n$. Dann ist $q(x) \in \Pi_{n-1}$ und somit $(q, p_n) = 0$.

Aber $q(x)p_n(x)$ hat keinen Vorzeichenwechsel in $]a,b[$ und

$$(q, p_n) = \int_a^b dx \ w(x)q(x)p_n(x) \neq 0$$

Widerspruch zu $k < n!$

$\rightarrow q(x)$ ist vom Grad n und $p_n(x)$ hat **n einfache Nullstellen** in $]a,b[$



Wir wählen jetzt als Stützstellen x_i die Nullstellen von $p_n(x)$ (n verschiedene Punkte!).

Dann kann man Gewichte finden, so dass für jedes Polynom $f(x) \in \Pi_{2n-1}$ exakt gilt:

$$\int_a^b dx w(x) f(x) = \sum_{i=0}^{n-1} \omega_i f(x_i)$$

Beweisidee

Wir zerlegen erst mit Polynomdivision $f(x) = p_n(x) q(x) + r(x)$ wobei $q, r \in \Pi_{n-1}$

Dann gilt für das Integral

$$\int_a^b dx w(x) f(x) = \underbrace{(p_n, q)}_{=0} + (r, p_0) = (r, p_0)$$

Entwickeln $r(x)$ als

$$r(x) = \sum_{j=0}^{n-1} \beta_j p_j(x)$$

Das Integral vereinfacht sich damit zu

$$\int_a^b dx w(x) f(x) = \sum_{j=0}^{n-1} \beta_j \underbrace{(p_j, p_0)}_{\propto \delta_{j0}} = \beta_0 (p_0, p_0)$$

Für die Nullstellen x_i von $p_n(x)$ findet man $f(x_i) = \underbrace{p_n(x_i)}_{=0} q(x_i) + r(x_i) = r(x_i) = \sum_{j=0}^{n-1} \beta_j p_j(x_i)$

oder als Gleichungssystem

$$\begin{pmatrix} f(x_0) \\ \vdots \\ f(x_{n-1}) \end{pmatrix} = \underbrace{\begin{pmatrix} p_0(x_0) & \cdots & p_j(x_0) & \cdots & p_{n-1}(x_0) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_0(x_{n-1}) & \cdots & p_j(x_{n-1}) & \cdots & p_{n-1}(x_{n-1}) \end{pmatrix}}_A \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{n-1} \end{pmatrix}$$

woraus man durch Invertieren

$$\int_a^b dx w(x) f(x) = \beta_0 (p_0, p_0) = \sum_{i=0}^{n-1} \underbrace{(p_0, p_0) [A^{-1}]_{0i}}_{\omega_i} f(x_i) = \sum_{i=0}^{n-1} \omega_i f(x_i)$$

erhält.



Man kann mit Hilfe der positiven Polynome $f(x) \equiv f_j(x) = \prod_{k=0}^{n-1} (x - x_k)^2$ zeigen,

dass die Gewichte positiv sind (Polynome vom Grad $2n - 2$!).

Beweisidee

$$\overbrace{\int_a^b dx w(x) f_j(x)}^{> 0} = \sum_{i=0}^{n-1} \omega_i \underbrace{f_j(x_i)}_{\propto \delta_{ij}} = \omega_j \overbrace{\prod_{k \neq j}^{n-1} (x_j - x_k)^2}^{> 0}$$

$$\implies \omega_j > 0 \quad \forall j$$

Diese Gewichte eignen sich auch um beliebige glatte Funktionen mit hoher Genauigkeit numerisch zu integrieren (nicht nur Polynome vom Grad $2n - 1$!):

In dem Fall ist die Relation nicht exakt sondern eine Approximation:

$$\int_a^b dx w(x) f(x) \approx \sum_{i=0}^{n-1} \omega_i f(x_i)$$

Der Verlauf der Gaußschen Integration für allgemeine $w(x) \geq 0$ und n Stützstellen:

Schritt 1

Konstruieren Sie $p_0, p_1(x), \dots, p_n(x)$ mithilfe der Rekursion

$$p_{i+1}(x) = (x - a_i) p_i(x) - b_i p_{i-1}(x) \quad \text{mit} \quad a_i = \frac{(xp_i, p_i)}{(p_i, p_i)} \quad b_i = \frac{(p_i, p_i)}{(p_{i-1}, p_{i-1})}$$

Skalarprodukt

wobei man mit $p_{-1} = 0$, $p_0 = 1$ und $b_0 = 1$ starten kann.

Schritt 2

Finden Sie die n Nullstellen x_0, x_1, \dots, x_{n-1} vom $p_n(x)$

Nutzen Sie das Newtonverfahren (z.B.)!

Schritt 3

Bestimmen Sie die Gewichte $\omega_0, \omega_1, \dots, \omega_{n-1}$ durch Lösen des Gleichungssystems

$$\underbrace{\begin{pmatrix} p_0(x_0) & p_0(x_1) & \cdots & p_0(x_{n-1}) \\ p_1(x_0) & p_1(x_1) & \cdots & p_1(x_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ p_{n-1}(x_0) & p_{n-1}(x_1) & \cdots & p_{n-1}(x_{n-1}) \end{pmatrix}}_{A^T} \begin{pmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_{n-1} \end{pmatrix} = \begin{pmatrix} (p_0, p_0) \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$\underbrace{(p_0, p_0)}_{\omega_i} \underbrace{[A^{-1}]_{0i}}$$

Annäherung des Integrals:

$$\int_a^b dx w(x)f(x) \approx \sum_{i=0}^{n-1} \omega_i f(x_i)$$

Verschiedene Varianten dieser Methode sind ausgearbeitet:

1. $w(x) = 1, a=-1, b=1 \rightarrow$ Legendre-Polynome (**Gauss-Legendre** Int.)
2. $w(x) = e^{-x^2}, a=-\infty, b=\infty \rightarrow$ Hermite-Polynome (**Gauss-Hermite** Int.)
3. $w(x) = \frac{1}{\sqrt{1-x^2}}, a=-1, b=1 \rightarrow$ Chebyshev-Polynome (**Gauss-Chebyshev** Int.)

. . . und andere. Sehen Sie, z.B., **Numerische Recipe** Kapitel 4.5

Falls sich das benötigte Integrationsintervall von den oben definierten unterscheidet. Ist eine einfache (lineare) **Transformation der Variablen** im Fall Gauss-Legendre hilfreich.

$$\int_a^b dx f(x) = \int_{-1}^1 dy \frac{b-a}{2} f\left(\frac{a+b}{2} + \frac{b-a}{2}y\right) \approx \sum_{i=0}^{n-1} \underbrace{\omega_i}_{\tilde{\omega}_i} \frac{b-a}{2} f\left(\underbrace{\frac{a+b}{2} + \frac{b-a}{2}y_i}_{x_i}\right)$$

definiert neue Gewichte
und Stützstellen

Es existieren Bibliotheks-Routinen, die Gewichte und Stützstellen zurückgeben!

Wir benutzen im folgenden die **GNU Scientific Library** (gsl).

```
/* Datei: beispiel-3.5.c   Datum: 21.4.2016 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

Gauss-Legendre aus GSL

```
/* GNU Scientific Library (gsl) Routinen erlauben Gauss-Legendre Punkte zu bestimmen */
#include <gsl/gsl_integration.h>
```

```
void gausslegendre(double a,double b,double *x,double *w,size_t n)
{ gsl_integration_glfixed_table *xwtable;
size_t i;
xwtable=gsl_integration_glfixed_table_alloc(n);
if(xwtable==NULL)
{
    printf("Problem with Gauss-Legendre\n");
    abort();
}
for(i=0;i<n;i++)
{
    gsl_integration_glfixed_point (a, b, i, &x[i], &w[i], xwtable);
}
gsl_integration_glfixed_table_free(xwtable);
}
```

```
/* Definition der zu integrierenden Funktion */
/* Definiere globale Variabel, um Parameter der Funktion festzulegen */
double aconst=0.5,bconst=1; /* Werte sind bei Programmstart vorgegeben */
```

```
double f(double x)
{
    return aconst/(bconst+x*x);
    /* ... auch hier kann ich globale Variablen benutzen */
}
```

GSL stellt Datentypen zur Verfügung

falls Problem: Programm abbrechen

**GSL gibt Stützstellen und Gewichte
nur einzeln heraus**

danach ist GSL Tabelle nicht mehr notwendig

**zu integrierende Funktion ist mit Hilfe
globaler Variablen definiert**

```

int main()
{
    double a,b;                      /* Intervallgrenzen */
    int i,n;                         /* Schleifenvariable, Anzahl der Stuetzstellen */
    double exact,diff,sum;           /* Variablen, um Ergebnis zu speichern */
    double *x,*w;                    /* Zeiger auf Speicherplaetze
printf("Bitte geben Sie a,b und n ein: "); /* Eingabe der
scanf("%lf %lf %d",&a,&b,&n);
x=(double *) malloc(n*sizeof(double)); /* Anzahl der
w=(double *) malloc(n*sizeof(double)); /* waehrend des Programmautes ! */
gausslegendre(a,b,x,w,n); /* uebergibt n = Anzahl der P
                           /* Adressen der mit new alloca
/* Gitterpunkte und Gewichte sind nun bei x und w gespeic
/* Diese kann man fuer beliebige Funktionen benutzen */
/* Legt jetzt die Parameter der Funktion durch Definition der globalen Variablen fest */
aconst=1.0;
bconst=1.0;
sum=0.0;                          /* Bestimmung eines Integrals mit den Gitter und Gewichten */
for(i=0;i<n;i++)
{
    sum+=f(x[i])*w[i]; /* "+" Operator summiert f(xi)*w(xi) auf Summe auf */
}
exact=atan(b)-atan(a)
diff=fabs(sum-exact);
printf("N      gauleg      exact      diff \n\n");
printf("%d      %15.6e      %15.6e      %15.6e \n",n,sum,exact,diff);
free(x); /* Speicherbereich wieder freigeben */
free(w); /* ("deallozieren") */
return 0;
}

```

/* Compilation auf cip Rechnern: gcc beispiel-3.5.c -lgsl -lblas -lm -o beispiel-3.5

Ergebnis: fuer a b n = 0.0 1.0 5

Bitte geben Sie a,b und n ein: 0 1 5

N gauleg exact diff

erinnert an Trapez

Eingabe für Integralgrenzen etc.

Speicher für Stützstellen und Gewichte

Aufruf der Routine die Gauss-Legendre Stützstellen und Gewichte festlegt

bestimme Parameter der Funktion f

Summe über Stützstellen: wie bei trapez

Ausgabe und Vergleich

Speicherplatz freigeben

Compilation erfordert zusätzliche Bibliotheken

