Einführung in die Programmierung mit C++

Modern Family: Nichtlineare Modelle

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Inhalt

Modern Family: Nichtlineare Modelle

Newtonverfahren

- Nichtlineare Gleichungen
- Nichtlineare Optimierung

Modern Family Varianten

- while
- do-while
- for
- for und break
- for und continue

Inhalt

Modern Family: Nichtlineare Modelle

Newtonverfahren
- Nichtlineare Gleichungen
- Nichtlineare Optimierung

Modern Family Varianten
- while
- do-while
- for
- for und break
- for und continue

Bisher haben sich Gretchen und Julia auf lineare (in p) Modelle konzentriert, z.B. $y = p \cdot x$ oder $y = p \cdot x^2$.

Aus der Optimalitätsbedingung ergab sich eine lineare Gleichung für p, welche leicht mittels Umstellen und Division gelöst werden konnte.

Für allgemeine nichtlineare Modelle ist das nicht mehr der Fall. Die resultierenden nichtlinearen Gleichungen haben oft keine symbolische Lösung. Eine Lösung muss numerisch approximiert werden, z.B. mithilfe des Verfahrens von Newton.

Inhalt

Das Newtonverfahren approximiert die Lösung der nichtlinearen Gleichung

$$y = f(x) = 0$$

für einen gebenen Startwert x iterativ mittels

$$x = x - \frac{f(x)}{f'(x)} .$$

→ Tafel

Die Iteration endet sobald das Konvergenzkriterium $f(x) \leq \epsilon$ für ein gegebenes $0 < \epsilon \ll 1$ erfüllt ist.

Konvergenz ist im Allgemeinen nicht garantiert.

Zur Lösung von

$$\min_{x \in R} f(x)$$

wird das Newtonverfahren auf die Optimalitätsbedingung erster Ordnung

$$f'(x) = 0$$

angewandt. Es ergibt sich die Iteration

$$x = x - \frac{f'(x)}{f''(x)}$$

für einen gebenen Startwert für x.

Ein lokales Minumum verlangt ausserdem nach Erfüllung der Optimalitätsbedingung zweiter Ordnung $f''(x) > 0$.

Inhalt

Für das in p nichtlineare Modell

$$y = (p \cdot x)^2$$

betrachten wir auf dem Newtonverfahren basierende Parameterschätzer unter Verwendung verschiedener Kontrollflussstrukturen:

- **while**
- **do−while**
- **for**
- **for−break**
- **for−continue**

Für die erste Variante präsentieren wir Lösungen unter Verwendung symbolischer sowie numerischer Ableitungen. Erstere werden für die verbleibenden Varianten verwendet.

Modern Family

Implementierung


```
1  template<typename T>
2  T model(T p, T x) { return pow(p*x,2); }
3
4  template<typename T>
5  T e(T p, T x1, T y1, T x2, T y2) {
6    return pow(model(p,x1)-y1,2)+pow(model(p,x2)-y2,2);
7  }
8
9  template<typename T>
10 T estimate(T x1, T y1, T x2, T y2);
11
12 int main() {
13   using namespace std;
14   double x1=0,x2=0,y1=0,y2=0;
15   cout << "x1="; cin >> x1; cout << "y1="; cin >> y1;
16   cout << "x2="; cin >> x2; cout << "y2="; cin >> y2;
17   cout << estimate(x1,y1,x2,y2) << endl;
18   return 0;
19 }
```

Modern Family

while (Symbolische Ableitungen)


```
1  template<typename T>
2  T estimate(T x1, T y1, T x2, T y2) {
3    T p=1, aux1=pow(x1,2), aux2=pow(x2,2);
4    T aux3=aux1*(pow(p*x1,2)-y1)+aux2*(pow(p*x2,2)-y2);
5    T dedp=4*p*aux3;
6    while (fabs(dedp)>1e-9) { // look here
7      T ddedpp=4*(aux3+2*pow(p,2)*(pow(x1,4)+pow(x2,4)));
8      p-=dedp/ddedpp;
9      aux3=aux1*(pow(p*x1,2)-y1)+aux2*(pow(p*x2,2)-y2);
10     dedp=4*p*aux3;
11   }
12   return p;
13 }
```

Modern Family

while (Numerische Ableitungen)


```
1  template<typename T>
2  T estimate(T x1, T y1, T x2, T y2) {
3    const T eps=std::numeric_limits<T>::epsilon();
4    T p=1;
5    T d=dedp(p,x1,y1,x2,y2);
6    while (fabs(d)>sqrt(eps)) { // look here
7      p-=d/ddedpp(p,x1,y1,x2,y2);
8      d=dedp(p,x1,y1,x2,y2);
9    }
10   return p;
11 }
```

wobei ...

Modern Family

while (Numerische Ableitungen)

...

```
1  template<typename T>
2  T dedp(T p, T x1, T y1, T x2, T y2) {
3    const T eps=std::numeric_limits<T>::epsilon();
4    const T h=(p!=0) ? sqrt(sqrt(eps))*fabs(p) : sqrt(sqrt(eps));
5    return (e(p+h,x1,y1,x2,y2)-e(p-h,x1,y1,x2,y2))/(2*h);
6  }
7
8  template<typename T>
9  T ddedpp(T p, T x1, T y1, T x2, T y2) {
10   const T eps=std::numeric_limits<T>::epsilon();
11   const T h=(p!=0) ? sqrt(sqrt(eps))*fabs(p) : sqrt(sqrt(eps));
12   return (dedp(p+h,x1,y1,x2,y2)-dedp(p-h,x1,y1,x2,y2))/(2*h);
13 }
```

```
1  template<typename T>
2  T estimate(T x1, T y1, T x2, T y2) {
3    T p=1, aux1=pow(x1,2), aux2=pow(x2,2), dedp;
4    do { // look here
5      T aux3=aux1*(pow(p*x1,2)-y1)+aux2*(pow(p*x2,2)-y2);
6      dedp=4*p*aux3;
7      T ddedpp=4*(aux3+2*pow(p,2)*(pow(x1,4)+pow(x2,4)));
8      p-=dedp/ddedpp;
9    } while (fabs(dedp)>1e-9); // look here
10   return p;
11 }
```

Modern Family

for


```
1  template<typename T>
2  T estimate(T x1, T y1, T x2, T y2) {
3    T p=1, aux1=pow(x1,2), aux2=pow(x2,2);
4    T aux3=aux1*(pow(p*x1,2)-y1)+aux2*(pow(p*x2,2)-y2);
5    T dedp=4*p*aux3;
6    for (;fabs(dedp)>1e-9;) { // look here
7      T ddedpp=4*(aux3+2*pow(p,2)*(pow(x1,4)+pow(x2,4)));
8      p-=dedp/ddedpp;
9      aux3=aux1*(pow(p*x1,2)-y1)+aux2*(pow(p*x2,2)-y2);
10     dedp=4*p*aux3;
11   }
12   return p;
13 }
```

Modern Family

for und break


```
1  template<typename T>
2  T estimate(T x1, T y1, T x2, T y2) {
3    T p=1, aux1=pow(x1,2), aux2=pow(x2,2), dedp;
4    for(;;) { // look here
5      T aux3=aux1*(pow(p*x1,2)-y1)+aux2*(pow(p*x2,2)-y2);
6      dedp=4*p*aux3;
7      if (fabs(dedp)<=1e-9) break; // look here
8      T ddedpp=4*(aux3+2*pow(p,2)*(pow(x1,4)+pow(x2,4)));
9      p-=dedp/ddedpp;
10   }
11   return p;
12 }
```

Modern Family

for und continue


```
1  template<typename T>
2  T estimate(T x1, T y1, T x2, T y2) {
3    T p=1, aux1=pow(x1,2), aux2=pow(x2,2), dedp;
4    for(bool c=true;c;) { // look here
5      T aux3=aux1*(pow(p*x1,2)-y1)+aux2*(pow(p*x2,2)-y2);
6      dedp=4*p*aux3;
7      if (fabs(dedp)<=1e-9) { c=false; continue; } // look here
8      T ddedpp=4*(aux3+2*pow(p,2)*(pow(x1,4)+pow(x2,4)));
9      p-=dedp/ddedpp;
10   }
11   return p;
12 }
```

AUFWACHEN!

Implementieren sie weitere korrekte Kontrollflussmuster (auch, wenn diese in der Realität evtl. nie zum Einsatz kommen werden ...).

Wach?

Schreiben sie niemals solchen Code ...

```
1  template<typename T>
2  T estimate(T x1, T y1, T x2, T y2) {
3    T p=1, aux1=pow(x1,2), aux2=pow(x2,2), dedp;
4    for(bool c=true;c;) {
5      T aux3=aux1*(pow(p*x1,2)-y1)+aux2*(pow(p*x2,2)-y2);
6      dedp=4*p*aux3;
7      char terminate=fabs(dedp)<=1e-9 ? 'Y' : 'N';
8      switch (terminate) {
9        default: break;
10       case 89: c=false;
11     }
12     if (!c) continue;
13     T ddedpp=4*(aux3+2*pow(p,2)*(pow(x1,4)+pow(x2,4)));
14     p-=dedp/ddedpp;
15   }
16   return p;
17 }
```

Zusammenfassung

Modern Family: Nichtlineare Modelle

Newtonverfahren

- Nichtlineare Gleichungen
- Nichtlineare Optimierung

Modern Family Varianten

- while
- do-while
- for
- for und break
- for und continue