Einführung in die Programmierung mit C++

Überladung: Fallstudien

Uwe Naumann

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

RWTH Aachen

Inhalt

Matrix-Vektor Arithmetik

Parametersensitivitätsanalyse

Modern Family

Outline

Matrix-Vektor Arithmetik

Parametersensitivitätsanalyse

Modern Family

- Für $\mathrm{u}, \mathrm{v}, \mathrm{w} \in \boldsymbol{R}^n$ und $a \in \boldsymbol{R}$ implementieren wir
  - Elementzugriff: $v_i,\ i = 0, \ldots, n-1$
  - Euklidische Norm: $||\mathrm{v}||_2 \equiv \sqrt{\sum_{i=0}^{n-1} v_i^2}$
  - Addition: $\mathrm{w} = \mathrm{u} + \mathrm{v} \ \Leftrightarrow\ w_i = u_i + v_i,\ i = 0, \ldots, n-1$
  - Skalierung: $\mathrm{w} = a * \mathrm{u} \ \Leftrightarrow\ w_i = a * u_i,\ i = 0, \ldots, n-1$
  - Skalarprodukt: $a = \mathrm{u} * \mathrm{v} \ \Leftrightarrow\ a = \sum_{i=0}^{n-1} u_i * v_i$
  - Ein-/Ausgabeströme
- Für $A \in \boldsymbol{R}^{m \times n}$, $\mathrm{u} \in \boldsymbol{R}^n$, $\mathrm{w} \in \boldsymbol{R}^m$, $B \in \boldsymbol{R}^{n \times p}$, und $C \in \boldsymbol{R}^{m \times p}$ implementieren wir zusätzlich
  - Matrix-Vektor Produkt: $\mathrm{w} = A * \mathrm{u} \ \Leftrightarrow\ w_i = A_{i,*} * \mathrm{u},\ i = 0, \ldots, n-1$ (Beachte: $A_{i,*} \in \boldsymbol{R}^n$)
  - Matrix-Matrix Produkt: $C = A * B \ \Leftrightarrow\ C_{i,j} = A_{i,*} * B_{*,j},\ i = 0, \ldots, n-1,$ $j = 0, \ldots, p-1$ (Beachte: $B_{*,j} \in \boldsymbol{R}^n$)
  - Transponierung: $B = A^T \ \Leftrightarrow\ B_{j,i} = A_{i,j}$ (Beachte: $p = m$, d.h. $B \in \boldsymbol{R}^{n \times m}$).

Matrix-Vektor Arithmetik

Diskussion des Quellcodes


```
1  /// static vector type and arithmetic
2  template<int N, typename T>
3  class vector {
4  protected:
5    T vals[N];
6  ...
7  };
8
9  /// static matrix type and matrix-matrix/vector arithmetic
10 template<int M, int N, typename T>
11 class matrix : public vector<M,vector<N,T>> {
12 using vector<M,vector<N,T>>::vals;
13 ...
14 };
```

Matrix-Vektor Arithmetik

Demo


```
1  int main() {
2    const int n=2,m=3;
3    vector<n,float> x; matrix<m,n,float> A; matrix<n,m,float> B;
4    for (size_t i=0,k=1;i<n;i++) {
5      x[i]=k++;
6      for (size_t j=0;j<m;j++) { A[j][i]=k++; B[i][j]=k++; }
7    }
8    std::cout << "A=" << A << std::endl;
9    std::cout << "B=" << B << std::endl;
10   std::cout << "x=" << x << std::endl;
11   std::cout << "A*x=" << A*x << std::endl;
12   std::cout << "A*B=" << A*B << std::endl;
13   std::cout << "A.transpose()=" << A.transpose() << std::endl;
14   return 0;
15 }
```

Die folgende Ausgabe wird generiert:

```
A=[ [ 2 9 ] [ 4 11 ] [ 6 13 ] ]
B=[ [ 3 5 7 ] [ 10 12 14 ] ]
x=[ 1 8 ]
A*x=[ 74 92 110 ]
A*B=[ [ 96 118 140 ] [ 122 152 182 ] [ 148 186 224 ] ]
A.transpose()=[ [ 2 4 6 ] [ 9 11 13 ] ]
```

AUFWACHEN!

Können folgende Ausdrücke erfolgreich übersetzt werden?

```
1 std::cout << "A*A.transpose()=" << A*A.transpose() << std::endl;
2 std::cout << "(A*B).transpose()=" << (A*B).transpose() << std::endl;
3 std::cout << "A*B.transpose()=" << A*B.transpose() << std::endl;
```

Falls ja, was wird dann für

```
A=[ [ 2 9 ] [ 4 11 ] [ 6 13 ] ]
B=[ [ 3 5 7 ] [ 10 12 14 ] ]
```

ausgegeben?

Wach?

Versuchte Übersetzung von

```
1 | std::cout << "A*B.transpose()=" << A*B.transpose() << std::endl;
```

resultiert in folgender Fehlermeldung:

```
error: no match for ‘operator*’
(operand types are ‘matrix<3, 2, float>’ and ‘matrix<3, 2, float>’)
  138 |   std::cout << "A*B.transpose()=" << A*B.transpose() << std::endl;
      |                                      ~~~~~~~~~~~~~~~~
```

Die beiden ersten Ausdrücke werden übersetzt und resultieren in der folgenden Ausgabe:

```
A*A.transpose()=[ [ 85 107 129 ] [ 107 137 167 ] [ 129 167 205 ] ]
(A*B).transpose()=[ [ 96 122 148 ] [ 118 152 186 ] [ 140 182 224 ] ]
```

Outline

Matrix-Vektor Arithmetik

Parametersensitivitätsanalyse

Modern Family

Für eine gegebene Implementierung einer univariaten skalaren Funktion

$$y = f(x) : \boldsymbol{R} \to \boldsymbol{R}$$

als ein differenzierbares C++ Programm möchten wir erste und zweite Ableitung

$$f'(x) = \frac{dy}{dx} \in \boldsymbol{R} \quad \text{und} \quad f''(x) = \frac{d^2y}{dx^2} \in \boldsymbol{R}$$

mittels Überladung berechnen und damit die Sensitivität von y (bzw. seiner ersten Ableitung) bezüglich kleinster Variationen in x quantifizieren.

Solche Variationen können aufgrund von Fehlern bei der Bestimmung (z.B. Beobachtung) des Wertes von x auftreten. Ein hoher Absolutwert von $f'(x)$ an einem gegebenen Punkt x kann z.B. als Indikator dafür interpretiert werden, dass man in diesem Teil des Definitionsbereichs von f den Wert von x mit besonders hoher Genauigkeit bestimmen sollte bevor man ihn zur numerischen Simulation verwendet.

Das gegebene differenzierbare C++ Programm sei als Sequenz von differenzierbaren Elementaroperationen formuliert, z.B. kann

```
1 template<typename T>
2 T f(const T& x) {
3   T z=x+x;
4   return sin(z)*x;
5 }
```

in

```
1   T v1=x+x;
2   T v2=sin(v1);
3   T v3=v2*x;
```

zerlegt werden. Formal

$$v_j = f_j(v_i)_{i \prec j}, \quad j = 1, \ldots, q,$$

wobei $v_0 = x$ und $y = v_q$. Im obigen Beispiel ist $q = 3$.

Das algorithmische Differenzieren solcher differenzierbaren Programme beruht auf den elementaren Differentiationsregeln und der Kettenregel der Differentiation

$$f(x) = f_q(f_{q-1}(\dots f_1(x)\dots))$$

$$\Downarrow$$

$$f'(x) = f'_q(f_{q-1}(\dots f_1(x)\dots)) * f'_{q-1}(\dots f_1(x)\dots) * f'_1(x)\ ,$$

z.B.

- $f(x) = x \Rightarrow f'(x) = 1$
- $f(x) = \sin(g(x)) \Rightarrow f'(x) = \cos(g(x)) * g'(x)$
- $f(x) = \cos(g(x)) \Rightarrow f'(x) = -\sin(g(x)) * g'(x)$
- $f(x) = -g(x) \Rightarrow f'(x) = -g'(x)$
- $f(x) = g(x) + h(x) \Rightarrow f'(x) = g'(x) + h'(x)$
- $f(x) = g(x) * h(x) \Rightarrow f'(x) = g'(x) * h(x) + g(x) * h'(x)$

Algorithmisches Differenzieren

Diskussion des Quellcodes


```
1  template<typename T>
2  struct AD {
3    T v=0,d=0;
4    AD(const T& v) : v(v), d(0) {}
5    void diff_wrt() { d=1; } // dx/dx=1
6  };
7
8  template<typename T>
9  AD<T> sin(const AD<T>& x) {
10   AD<T> y(T(0)); y.v=sin(x.v); y.d=cos(x.v)*x.d;
11   return y;
12 }
13
14 template<typename T>
15 AD<T> operator*(const AD<T>& x1, const AD<T>& x2) {
16   AD<T> y(T(0)); y.v=x1.v*x2.v; y.d=x1.d*x2.v+x1.v*x2.d;
17   return y;
18 }
```

Für $x = 1$ und

```
1  template<typename T>
2  T f(const T& x) {
3    T z=x+x;
4    return sin(z)*x;
5  }
```

erhalten wir

$$f(x) = 0.909297$$
$$f'(x) = 0.0770038$$
$$f''(x) = -5.30178$$

validiert mittels finiter Differenzen.

Für $x = 1$ und

```
1 template<typename T>
2 T f(T x) {
3   for (size_t i=0;i<42;i++) x=sin(cos(x*x)+x)*x;
4   return -x;
5 }
```

erhalten wir

$$f(x) = -0.986291$$
$$f'(x) = -0.488053$$
$$f''(x) = 8.23238$$

validiert mittels finiter Differenzen.

Bei STCE beschäftigen wir uns intensiv mit algorithmischem Differenzieren. Mehr dazu in unseren entsprechenden Spezialvorlesungen.

AUFWACHEN!


```
1 template<typename T>
2 T f(T x) {
3   x=T(1);
4   return x*x;
5 }
```

$$f(x) = ?$$
$$f'(x) = ?$$
$$f''(x) = ?$$

Wach?

```
1 template<typename T>
2 T f(T x) {
3   x=T(1);
4   return x*x;
5 }
```

$$f(x) = 1$$
$$f'(x) = 0$$
$$f''(x) = 0$$

Outline

Matrix-Vektor Arithmetik

Parametersensitivitätsanalyse

Modern Family

Im Kontext der Modern Family Fallstudie könnte man an der Sensitivität des Parameterschätzers bezüglich der beobachteten Werte interessiert sein.

Ein hoher Absolutwert der Ableitung des geschätzten Parameters p bezüglich der zuletzt getätigten Beobachtung (z.B. von y für gegebenes x) impliziert einen starken Einfluss von Fehlern in der Beobachtung auf den geschätzten Wert von p. Die Damen sollte demnach in solchen Situationen ganz genau hinschauen ...

Diskussion des Quellcodes


```
1  #include "data.hpp"
2  #include "linear_model.hpp"
3  #include "nonlinear_model.hpp"
4  #include "AD.hpp"
5
6  int main(int argc, char *argv[]) {
7    assert(argc==3);
8    using T=AD<double>;
9    data<T> d(argv[1]); linear_model<T> m(d,argv[2]);
10   T x,y;
11   std::cout << "new observation:" << std::endl;
12   std::cout << "x="; std::cin >> x;
13   std::cout << "y="; std::cin >> y;
14   y.diff_wrt("y");
15   d.add_observation(x,y);
16   m.calibrate();
17   std::cout << "sensitivity of p: " << derivative(m.parameter()) << std::endl;
18   ...
19   return 0;
20 }
```

Für das nichtlineare (in p) Modell $y = (p \cdot x)^2$ und Startwert $p = 1$ erhalten wir nach einer Beobachtung $(x = 1, y = 2)$

$$p(x, y) = 1.41413$$
$$\frac{dp}{dy} = 0.353532 \ .$$

Eine zweite Beobachtung $(x = 2, y = 3)$ ergibt

$$p(x, y) = 0.90743$$
$$\frac{dp}{dy} = 0.129633 \ ,$$

etc.

Ausserdem kann man auch die Ableitung eines simulierten $\tilde{y}$ für zuvor geschätztes p und gegebenes x betrachten.

```
1  ...
2
3  int main(int argc, char *argv[]) {
4    assert(argc==3);
5    using T=AD<double>;
6    data<T> d(argv[1]); nonlinear_model<T> m(d,argv[2]);
7    T x,y;
8    ...
9    y.diff_wrt("y");
10   ...
11   std::cout << "new simulation:" << std::endl;
12   std::cout << "x="; std::cin >> x;
13   y=m.evaluate(x);
14   std::cout << "y=" << y << std::endl;
15   std::cout << "sensitivity of y: " << derivative(y) << std::endl;
16   return 0;
17 }
```

Für $y = (p \cdot x)^2$ und Startwert $p = 1$ erhalten wir nach einer Beobachtung $(x = 1, y = 2)$

$$\tilde{y} = y(p = 1.41413, x = 2) = 7.99902$$
$$\frac{d\tilde{y}}{dy} = 3.99951 \ .$$

Die zweite Beobachtung $(x = 2, y = 3)$ ergibt z.B.

$$\tilde{y} = y(p = 0.90743, x = 3) = 7.41086$$
$$\frac{d\tilde{y}}{dy} = 2.11739 \ .$$

Zusammenfassung

Matrix-Vektor Arithmetik

Parametersensitivitätsanalyse

Modern Family