Einführung in die Programmierung mit C++

Modern Family Tour: Richtung Realität

Uwe Naumann

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

RWTH Aachen

Inhalt

Parameterschätzer für mehrere Beobachtungen

Eingabe aus / Ausgabe in Textdatei

Fehlersuche mit den gdb Debugger

Vorteil defensive Programmierung

Inhalt

Schnell leitet Julia aus

$$f(p, \mathrm{x}, \mathrm{y}) = \sum_{i=1}^{m} (p \cdot x_i - y_i)^2 \quad \rightarrow \min$$

eine allgemeine Form des Parameterschätzers her:

Aus

$$\frac{df}{dp}(p, \mathrm{x}, \mathrm{y}) = \sum_{i=1}^{m} 2 \cdot x_i \cdot (p \cdot x_i - y_i) = 0$$

folgt unmittelbar

$$p = \frac{\mathrm{x}^T \cdot \mathrm{y}}{\mathrm{x}^T \cdot \mathrm{x}} = \frac{\sum_{i=1}^{m} x_i \cdot y_i}{\sum_{i=1}^{m} x_i^2}$$

und damit eine Formel, mit welcher man für eine beliebige Zahl von Beobachtungen (x_i, y_i), $i = 1, \ldots, m$, den bestmöglichen Parameter p bestimmen kann. Diese implementiert sie auch gleich ...

Parameterschätzer für mehrere Beobachtungen I


```
1  // naumann@stce.rwth-aachen.de
2
3  #include <iostream>
4  #include <cassert>
5  #include <cmath>
6  #include <vector> // stdlib vectors
7
8  /* takes vector of observations y at corresponding positions x
9     and returns optimal estimate for free parameter p of model y=p*x */
10 float estimate(std::vector<float> x, std::vector<float> y) {
11   assert(x.size()==y.size());
12   float xTy=0,xTx=0;
13   for (size_t /* unsigned integer type */i=0;i<x.size();i++) {
14     xTy+=x[i]*y[i];
15     xTx+=pow(x[i],2);
16   }
17   assert(xTx!=0);
18   return xTy/xTx;
19 }
20
```

Modern Family

Parameterschätzer für mehrere Beobachtungen II


```
21 int main() {
22   int m=0; // size of data
23   std::cout << "m="; std::cin >> m;
24   std::vector<float> x(m,0),y(m,0); // initialized vectors to hold data
25   for (int i=0;i<m;i++) {
26     std::cout << "x[" << i << "]="; std::cin >> x[i];
27     std::cout << "y[" << i << "]="; std::cin >> y[i];
28   }
29   float p=estimate(x,y);
30   std::cout << "p=" << p << std::endl;
31   return 0;
32 }
```

→ Live: Tour_MF006.cpp

Beachte

- Vektor von Elementen generischen Typs: std::vector
- Datentyp für nichtnegative ganze Zahlen: size_t
- Kontrollflussstruktur: **for**-Schleife

Da

$$p \cdot x = 1.6 \cdot 1.5 = 2.4$$

begibt sich Julia gutgelaunt nach $(1.5, 2.4)$ nur um Faust und Romeo von $(1.5, 1)$ winken zu sehen. “Nicht mein Tag heute ...”

Tapfer beschliessen sie und Gretchen, mit der Assimilation von Beobachtungsdaten in ihr Modell fortzufahren.

Aus den ersten drei Beobachtungen folgt $p = 1.31034$.

Daten in `data.in`

```
3
2 3
1 2
1.5 1
```

und

```
:-) ./Tour_MF006.exe < data.in
```

liefert (leicht derangierte) Ausgabe

```
m=x[0]=y[0]=x[1]=y[1]=x[2]=y[2]=p=1.31034
```

1. Daten in `data.plot` (=`data.in` ohne erste Zeile `m`)

2. `gnuplot` script `plot.gp`

```
set terminal pdf
set output "plot.pdf"
plot [0:3] "data.plot", 1.31034*x
```

3. Ausführen von `gnuplot` script

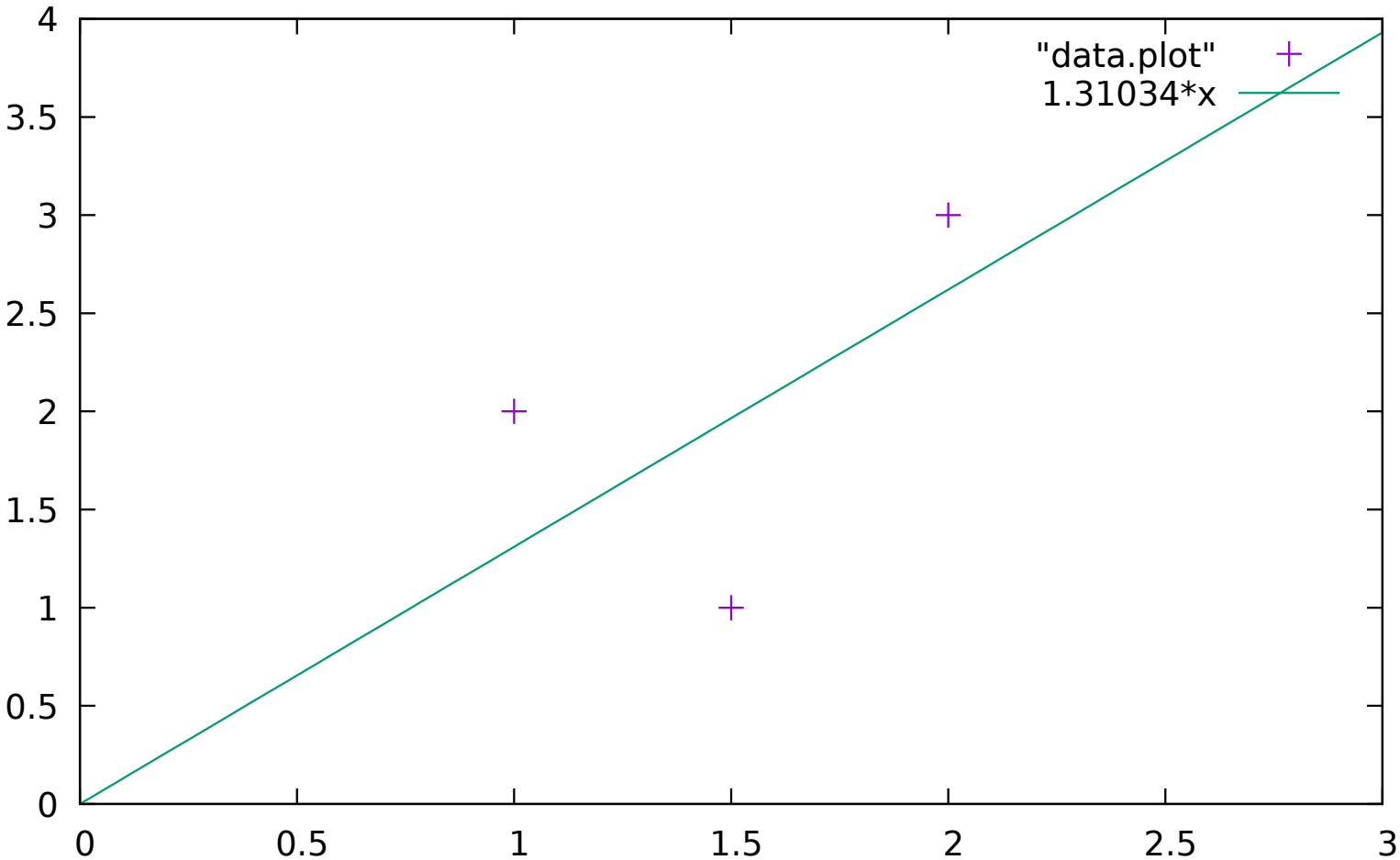
```
:-) gnuplot plot.gp
:-)
```

4. Betrachten von Datei `plot.pdf`, z.B.

```
:-) opera plot.pdf
```

Parameterschätzer für mehrere Beobachtungen

Visualisierung mit `gnuplot`



Modifizieren sie obiges Programm so, dass $y = p \cdot x^2$ als alternatives Modell verwendet wird.

Wach?

$$f(p, x, y) = \sum_{i=1}^{m} (p \cdot x_i^2 - y_i)^2 \quad \Rightarrow$$

$$\frac{df}{dp}(p, x, y) = \sum_{i=1}^{m} 2 \cdot x_i^2 \cdot (p \cdot x_i^2 - y_i) = 0 \quad \Rightarrow$$

$$p = \frac{\sum_{i=1}^{m} x_i^2 \cdot y_i}{\sum_{i=1}^{m} x_i^4}$$

wobei

$$\frac{d^2 f}{dp^2}(p, x, y) = \sum_{i=1}^{m} 2 \cdot x_i^4 \geq 0 \; .$$

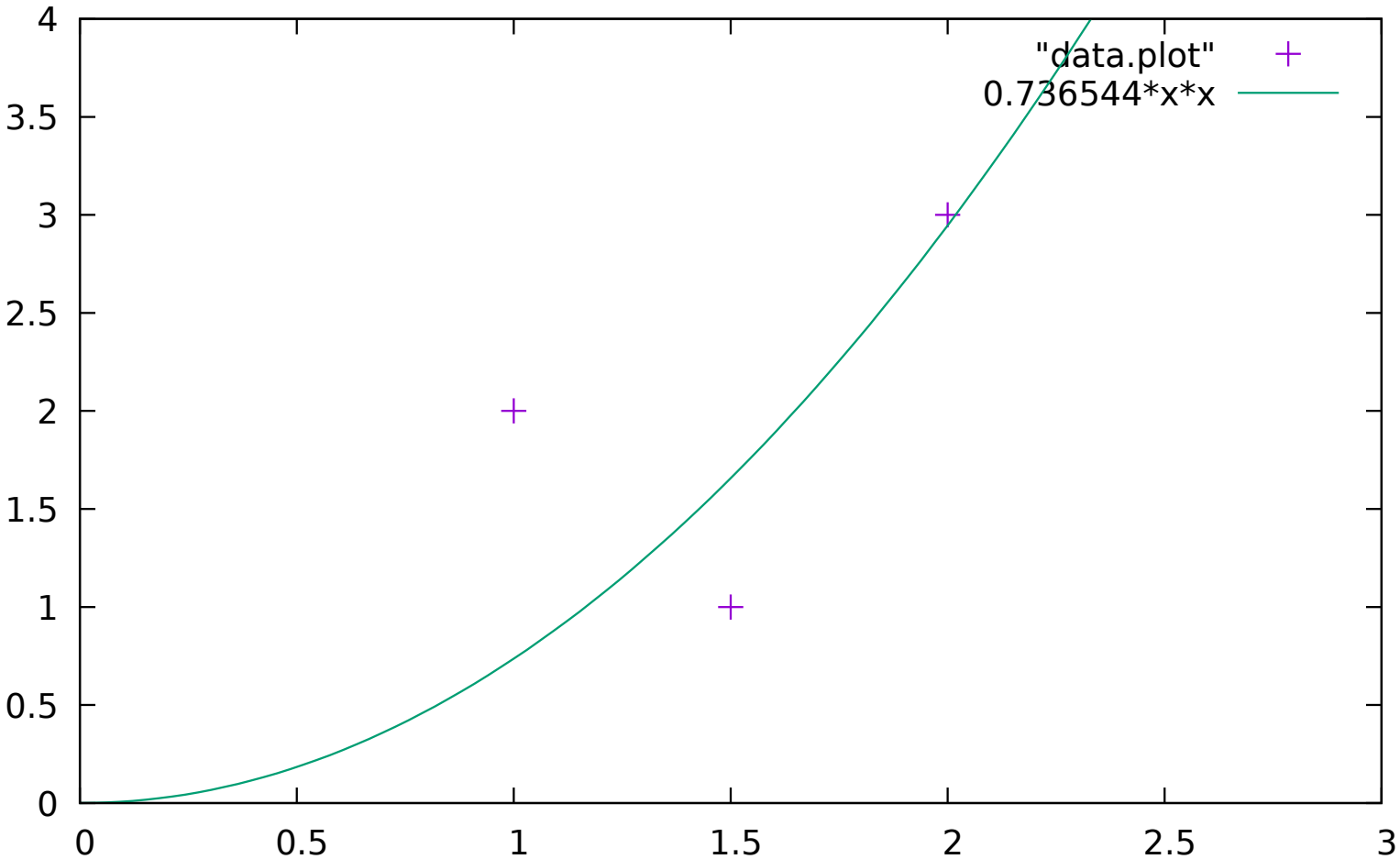
Wach?

```
1  /* takes vector of observations y at corresponding positions x
2      and returns optimal estimate for free parameter p of model y=p*x^2 */
3  float estimate(std::vector<float> x, std::vector<float> y) {
4    assert(x.size()==y.size());
5    float x2Ty=0,x2Tx2=0;
6    for (size_t i=0;i<x.size();i++) {
7      x2Ty+=pow(x[i],2)*y[i];
8      x2Tx2+=pow(x[i],4);
9    }
10   assert(x2Tx2!=0);
11   return x2Ty/x2Tx2;
12 }
```

→ Live: Tour_MF006_1.cpp

Wach?

```
:-) ./Tour_MF006_1.exe < data.in
m=x[0]=y[0]=x[1]=y[1]=x[2]=y[2]=p=0.736544
```



Da

$$p \cdot x = 1.6 \cdot 1.5 = 2.4$$

begibt sich Julia gutgelaunt nach $(1.5, 2.4)$ nur um Faust und Romeo von $(1.5, 1)$ winken zu sehen. “Nicht mein Tag heute ...”

Tapfer beschliessen sie und Gretchen, mit der Assimilation von Beobachtungsdaten in ihr Modell fortzufahren.

Aus den ersten drei Beobachtungen folgt $p = 1.31034$.

Wäre es nicht angenehm, wenn eine Datei mit den Messdaten automatisch aktualisiert würde ...

Inhalt

Parameterschätzer I


```
1  // naumann@stce.rwth-aachen.de
2
3  #include <iostream>
4  #include <cassert>
5  #include <cmath>
6  #include <vector>
7  #include <fstream> // file i/o
8
9  float estimate(std::vector<float> x, std::vector<float> y) {
10   float xTy=0,xTx=0;
11   for (size_t i=0;i<x.size();i++) { xTy+=x[i]*y[i]; xTx+=pow(x[i],2); }
12   assert(xTx!=0); return xTy/xTx;
13 }
14
15 int main() {
16   std::string data_file_name; // string
17   std::cout << "name of data file="; std::cin >> data_file_name;
18   std::ifstream data_file_read; // enables read access to file
19   data_file_read.open(data_file_name); // open data file for reading
20   float p=0; data_file_read >> p; // read value of free parameter from file
21   size_t m=0; data_file_read >> m; // read integer size of data from file
```

```
22    std::vector<float> x(m+1),y(m+1); // given data and single new data item
23    for (size_t i=0;i<m;i++) {
24      data_file_read >> x[i]; assert(x[i]>0);
25      data_file_read >> y[i]; assert(y[i]>0);
26    }
27    data_file_read.close(); // close data file
28
29    std::cout << "new data item:" << std::endl;
30    std::cout << "x="; std::cin >> x[m]; std::cout << "y="; std::cin >> y[m];
31    p=estimate(x,y);
32    std::cout << "p=" << p << std::endl;
33
34    std::ofstream data_file_write; // enables write access to file
35    data_file_write.open(data_file_name); // open data file for overwriting
36    data_file_write << p << std::endl << m+1 << std::endl; // update data file
37    for (size_t i=0;i<=m;i++) data_file_write << x[i] << " " << y[i] << std::endl;
38    data_file_write.close(); // close data file
39    return 0;
40  }
```

→ Live: Tour_MF007.cpp

1. Daten in `data.txt` $(p, m, (x_i, y_i))$

```
1.6
2
2 3
1 2
```

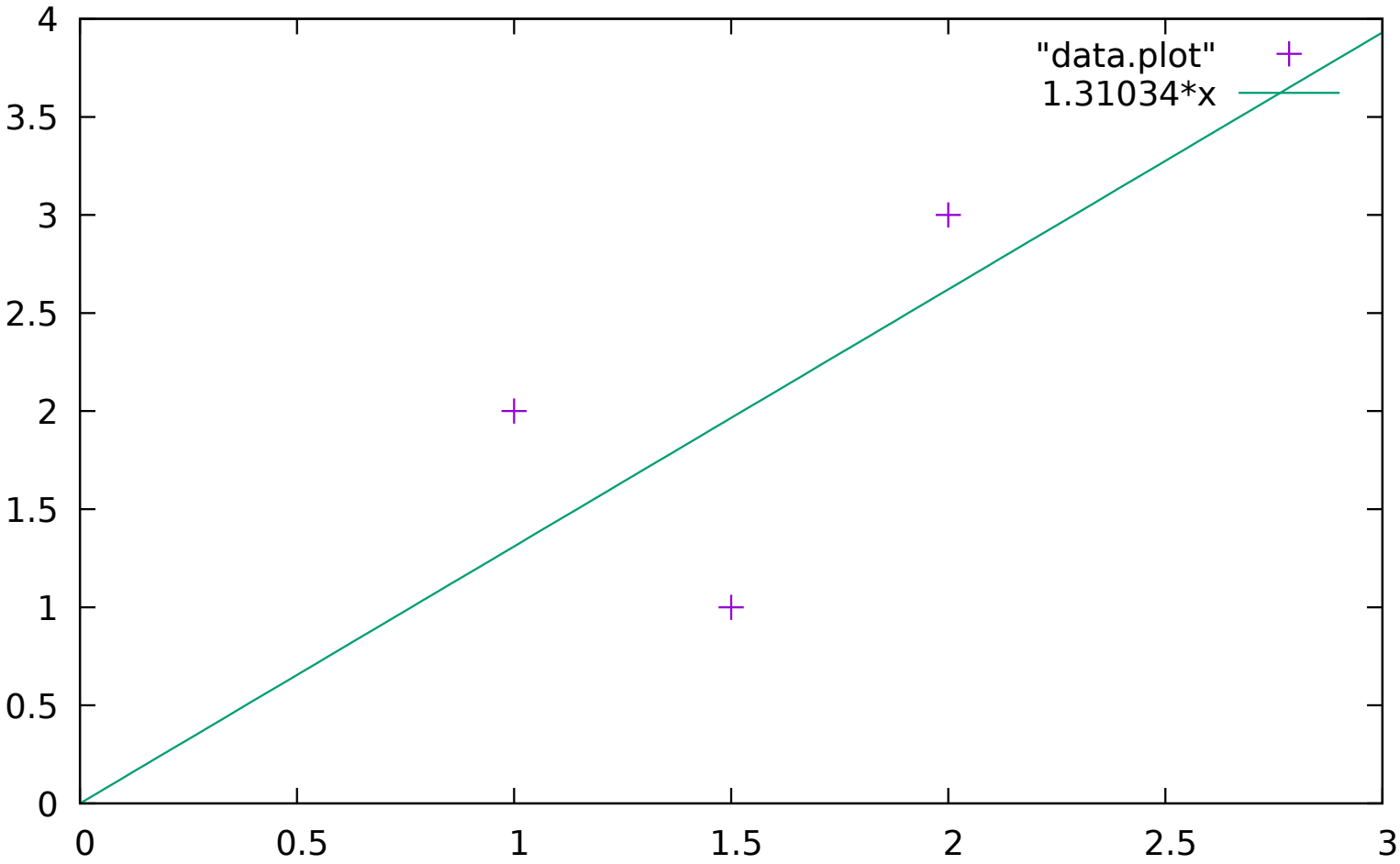
2. Programm ausführen und Dialog folgen

```
:-) ./Tour_MF007.exe
name of data file=data.txt
new data item:
x=1.5
y=1
p=1.31034
```

3. neue Daten in `data.txt`

```
1.31034
3
2 3
1 2
1.5 1
```

→ Live: Sequenz von Anwendungen


"data.plot"
1.31034*x
4
3.5
3
2.5
2
1.5
1
0.5
0
0
0.5
1
1.5
2
2.5
3

Inhalt

Nach Elimination aller Zusicherungen ...

```
3  ...
4  // #include <cassert> // offensive on purpose
5  ...
```

... fügen wir einen “kleinen” Fehler in das Programm ein:

```
40 ...
41   for (size_t i=0;i</*=*/m;i++)
42     data_file_write << x[i] << " " << y[i] << std::endl;
43 ...
```

→ Live: Tour_MF007_offensive_error.cpp

Die Konsequenz ist weniger “klein” ...

Ausgehend von data.txt mit dem Inhalt

```
1 0
2 0
```

($p = m = 0$) fügen wir die Beobachtung $(1, 1)$ hinzu, woraus $p = 1$ folgt.

In einer zweiten Iteration fügen wir die Beobachtung $(2, 3)$ hinzu, woraus $p = 1.4$ folgen müsste. Jedoch wird der fehlerhafte Wert $p = 1.5$ berechnet.

Wie den Fehler finden?

- wiederholtes Durchdenken des Quellcodes
- Inspektion der Daten
- zusätzliches Einfügen von Ausgaben in den Quellcode
- defensives Programmieren
- schrittweise Inspektion des Programmablaufs im Debugger

```
(gdb) break main
Breakpoint 1 at Tour_MF007_offensive_error.cpp, line 16.
(gdb) run
Starting program: Tour_MF007_offensive_error.exe
Breakpoint 1, main () at Tour_MF007_offensive_error.cpp:16
16 int main() {
(gdb) n
17 std::string data_file_name;
(gdb)
18 std::cout << "name of data file="; std::cin >> data_file_name;
(gdb)
name of data file=data.txt
19 std::ifstream data_file_read;
(gdb) p data_file_name
$1 = "data.txt"
(gdb) n
20 data_file_read.open(data_file_name);
(gdb)
21 float p=0;
(gdb)
```

```
22 data_file_read >> p;
(gdb)
23 size_t m=0;
(gdb) p p
$2 = 1
(gdb) n
24 data_file_read >> m;
(gdb)
25 std::vector<float> x(m+1),y(m+1);
(gdb) p m
$3 = 1
(gdb) n
26 for (size_t i=0;i<m;i++) {
(gdb)
27 data_file_read >> x[i]; // assert(x[i]>0);
(gdb)
28 data_file_read >> y[i]; // assert(y[i]>0);
(gdb)
26 for (size_t i=0;i<m;i++) {
(gdb)
30 data_file_read.close();
```

```
42 (gdb) p x
43 $4 = std::vector of length 2, capacity 2 = {0, 0}
44 (gdb) p y
45 $5 = std::vector of length 2, capacity 2 = {0, 0}
46 (gdb) n
47 32 std::cout << "new data item:" << std::endl;
48 (gdb)
49 new data item:
50 33 std::cout << "x="; std::cin >> x[m];
51 (gdb)
52 x=2
53 34 std::cout << "y="; std::cin >> y[m];
54 (gdb)
55 y=3
56 35 p=estimate(x,y);
57 (gdb)
58 36 std::cout << "p=" << p << std::endl;
59 (gdb)
60 p=1.5
61 38 std::ofstream data_file_write;
62 (gdb)
```

```
63 | 39 data_file_write.open(data_file_name);
64 | (gdb) p data_file_name
65 | $6 = "data.txt"
66 | (gdb) n
67 | 40 data_file_write << p << std::endl << m+1 << std::endl;
68 | (gdb) p p
69 | $7 = 1.5
70 | (gdb) p m+1
71 | $8 = 2
72 | (gdb) n
73 | 41 for (size_t i=0;i<m;i++)
74 | (gdb)
75 | 42 data_file_write << x[i] << " " << y[i] << std::endl;
76 | (gdb)
77 | 41 for (size_t i=0;i<m;i++)
78 | (gdb)
79 | 43 data_file_write.close();
80 | (gdb)
81 | 44 return 0;
82 | (gdb) q
```

Inhalt

Der Debugger ist dein Freund!

Siehe `www.gnu.org/software/gdb` sowie zukünftige Vorlesungen, Globalübungen und Tutorien für weitere Details zu `gdb`.

Übrigens: Defensive Programmierung hätte uns einige "Schmerzen" ersparen können.

→ Live: Tour_MF007_defensive_error.cpp

Modern Family

... und sie lebten glücklich ...

... und es kamen die Zweifel ...

Zusammenfassung

Parameterschätzer für mehrere Beobachtungen

Eingabe aus / Ausgabe in Textdatei

Fehlersuche mit den `gdb` Debugger

Vorteil defensive Programmierung