Modern Family

Regression in *R*

Uwe Naumann

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

RWTH Aachen University

Contents

Outline

Objective

- Introduction to sample code approaching the Modern Family example as a scalar regression problem.

Learning Outcomes

- You will understand
  - the formulation of the Modern Family example as a scalar regression problem
  - solution of the linear problem using normal equations, Givens and Housholder methods.
  - solution of the nonlinear problem as a sequence of linear problems using normal equations, Givens and Housholder methods.
- You will be able to
  - run the sample code
  - compare the results

Outline

For $y = f(p, x) = p \cdot x$

$$E = E(p, \mathbf{x}, \mathbf{y}) = \sum_{i=0}^{m-1} (f(p, x_i) - y_i)^2 = \sum_{i=0}^{m-1} (x_i \cdot p - y_i)^2 = \|p \cdot \mathbf{x} - \mathbf{y}\|_2^2$$

and hence

$$E' = \frac{dE}{dp}(p, \mathbf{x}, \mathbf{y}) = 2 \cdot \sum_{i=0}^{m-1} (x_i \cdot p - y_i) \cdot x_i \quad (= 0?)$$

and

$$E'' = \frac{d^2E}{dp^2}(p, \mathbf{x}, \mathbf{y}) = 2 \cdot \sum_{i=0}^{m-1} x_i^2 \quad (> 0?)$$

Solution of

$$\mathbf{x} \cdot p \approx \mathbf{y}$$

does not require derivatives of model.

Validation of first- and second-order optimality conditions

$$E' = 0 \quad \text{and} \quad E'' > 0$$

requires first and second derivatives $E', E'' \in \boldsymbol{R}$ of the error E.

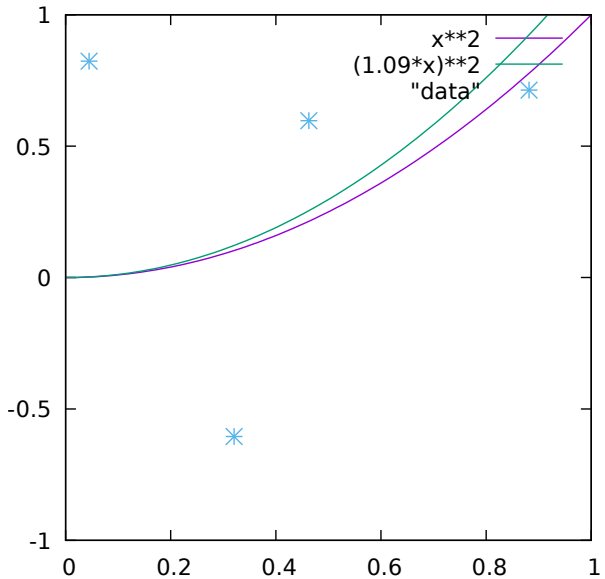
```
1  using namespace Eigen; // and trading const for space ...
2
3  template<typename T, int M>
4  T dEdp(T& p, Matrix<T,M,1> &x, <T,M,1> &y) {
5    return p*x.dot(x)-x.dot(y);
6  }
7
8  template<typename T, int M>
9  T ddEdpp(Matrix<T,M,1> &x) {
10   return x.dot(x);
11 }
```

```
1  using namespace Eigen; // and trading const for space ...
2
3  template<typename T, int M>
4  T dEdp(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
5    T dp=(p==0) ? sqrt(sqrt(std::numeric_limits<T>::epsilon()))
6                : sqrt(sqrt(std::numeric_limits<T>::epsilon()))*fabs(p);
7    return (E(p+dp,x,y)-E(p-dp,x,y))/(2*dp);
8  }
9
10 template<typename T, int M>
11 T ddEdpp(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
12   T dp=(p==0) ? sqrt(sqrt(std::numeric_limits<T>::epsilon()))
13               : sqrt(sqrt(std::numeric_limits<T>::epsilon()))*fabs(p);
14   return (dEdp(p+dp,x,y)-dEdp(p-dp,x,y))/(2*dp);
15 }
```

```
1  using namespace Eigen; // and trading const for space ...
2
3  template<typename TA, typename TP, int M>
4  TA dEdp(TA &p, Matrix<TP,M,1> &x, Matrix<TP,M,1> &y) {
5    using DCO_T=typename dco::gt1s<TA>::type;
6    DCO_T p_t=p;
7    dco::derivative(p_t)=1;
8    return dco::derivative(E(p_t,x,y));
9  }
10
11 template<typename T, int M>
12 T ddEdpp(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
13   using DCO_T=typename dco::gt1s<T>::type;
14   DCO_T p_t=p;
15   dco::derivative(p_t)=1;
16   return dco::derivative(dEdp(p_t,x,y));
17 }
```

Inspection of source code and live demo.

```
./main.exe 3
x^T= 0.462911 0.0446199 0.320581
y^T= 0.59688 0.823295 -0.604897
p=0.373357
E'=0
E''=0.319049
```



Outline

For $y = f(p, x) = (p \cdot x)^2$

$$E = E(p, \mathbf{x}, \mathbf{y}) = \sum_{i=0}^{m-1} (f(p, x_i) - y_i)^2 = \sum_{i=0}^{m-1} \left((p \cdot x_i)^2 - y_i\right)^2$$

and hence

$$E' = \frac{dE}{dp}(p, \mathbf{x}, \mathbf{y}) = 4 \cdot p \cdot \sum_{i=0}^{m-1} p^2 \cdot x_i^4 - x_i^2 \cdot y_i \quad (= 0?)$$

and

$$E'' = \frac{d^2E}{dp^2}(p, \mathbf{x}, \mathbf{y}) = 4 \cdot \sum_{i=0}^{m-1} 3 \cdot p^2 \cdot x_i^4 - x_i^2 \cdot y_i \quad (> 0?)$$

Solution of

$$F'(p, \mathbf{x}, \mathbf{y}) \cdot \Delta p \approx -F(p, \mathbf{x}, \mathbf{y})$$

requires $F'(p, \mathbf{x}, \mathbf{y}) \in R^m$.

Validation of first- and second-order optimality conditions

$$E' = 0 \quad \text{and} \quad E'' > 0$$

requires $E', E'' \in \boldsymbol{R}$.

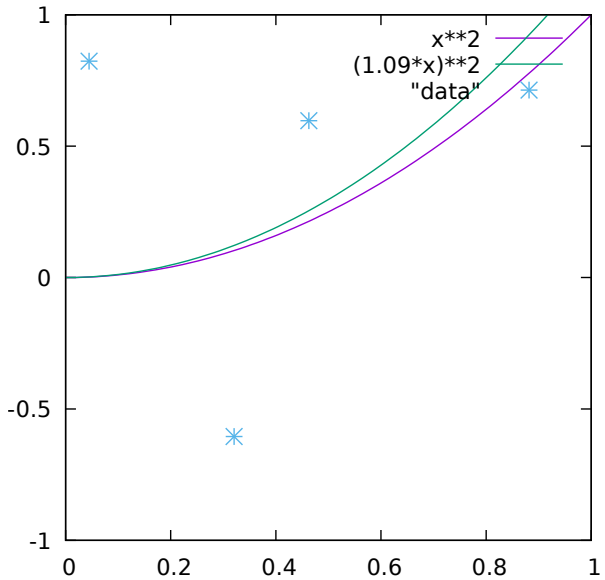
```
1  using namespace Eigen; // and trading const for space ...
2
3  template<typename T, int M>
4  Matrix<T,M,1> dFdp(T &p, Matrix<T,M,1> &x) {
5    return 2*p*x.cwiseProduct(x);
6  }
7
8  template<typename T, int M>
9  T dEdp(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
10   Matrix<T,M,1> xx=x.cwiseProduct(x);
11   return 4*p*(p*p*xx.dot(xx)-xx.dot(y));
12 }
13
14 template<typename T, int M>
15 T ddEdpp(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
16   Matrix<T,M,1> xx=x.cwiseProduct(x);
17   return 4*(3*p*p*xx.dot(xx)-xx.dot(y));
18 }
```

```
1  using namespace Eigen; // and trading const for space ...
2
3  template<typename T, int M>
4  Matrix<T,M,1> F(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
5    return p*p*x.cwiseProduct(x)-y;
6  }
7
8  template<typename T, int M>
9  Matrix<T,M,1> dFdp(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
10   T dp=(p==0) ? sqrt(std::numeric_limits<T>::epsilon())
11               : sqrt(std::numeric_limits<T>::epsilon())*fabs(p);
12   return (F(p+dp,x,y)-F(p,x,y))/dp;
13 }
14
15 // dEdp and ddEdpp as in linear case using ...
16
17 template<typename T, int M>
18 T E(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
19   return F(p,x,y).squaredNorm();
20 }
```

```
1  using namespace Eigen; // and trading const for space ...
2
3  template<typename TA, typename TP, int M>
4  Matrix<TA,M,1> F(TA &p, Matrix<TP,M,1> &x, Matrix<TP,M,1> &y);
5
6  template<typename T, int M>
7  Matrix<T,M,1> dFdp(T &p, Matrix<T,M,1> &x, Matrix<T,M,1> &y) {
8    using DCO_T=typename dco::gt1s<T>::type;
9    DCO_T p_t=p;
10   dco::derivative(p_t)=1;
11   Matrix<DCO_T,M,1> r_t=F(p_t,x,y);
12   auto m=x.size();
13   Matrix<T,M,1> drdp(m);
14   for (auto i=0;i<m;i++) drdp(i)=dco::derivative(r_t(i));
15   return drdp;
16 }
17
18 // dEdp and ddEdpp as in linear case using ...
19
20 template<typename TA, typename TP, int M>
21 TA E(TA &p, Matrix<TP,M,1> &x, Matrix<TP,M,1> &y);
```

Inspection of source code and live demo.

```
1 ./main.exe 3
2 x^T= 0.462911 0.0446199 0.320581
3 y^T= 0.59688 0.823295 -0.604897
4 p=1.09217
5 E'=4.4334e-06
6 E''=0.539019
```



Outline

Summary

- Introduction to sample code approaching the Modern Family example as a scalar regression problem.

Next Steps

- Play with the sample code.
- Continue the course to find out more ...