Modern Family

Regression in $\boldsymbol{R}^n$

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 nonscalar regression problem.

Learning Outcomes

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

Outline

For given data $(X = (\mathbf{x}_i) = (x_{i,j}), \mathbf{y}) \in \boldsymbol{R}^{m \times n} \times \boldsymbol{R}^m$ to be fitted by a linear model $y = f(\mathbf{p}, \mathbf{x}) = \mathbf{x}^T \cdot \mathbf{p}$ we minimize the error function

$$E = \sum_{i=0}^{m-1} (f(\mathbf{p}, \mathbf{x}_i^T) - y_i)^2 = \sum_{i=0}^{m-1} \left(\sum_{j=0}^{n-1} x_{i,j} \cdot p_j - y_i \right)^2 = \|X \cdot \mathbf{p} - \mathbf{y}\|_2^2$$

requiring

$$E' = \frac{dE}{d\mathbf{p}} = 2 \cdot \sum_{i=0}^{m-1} (\mathbf{x}_i \cdot \mathbf{p} - y_i) \cdot \mathbf{x}_i = 0$$

and a positive definite Hessian

$$E'' = \frac{d^2 E}{d\mathbf{p}^2} = 2 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot \mathbf{x}_i \ .$$

Validation of first- and second-order optimality conditions requires first and second derivatives of the error function

$$E = \|X \cdot \mathbf{p} - \mathbf{y}\|_2^2 = \sum_{i=0}^{m-1} \left(\sum_{j=0}^{n-1} x_{i,j} \cdot p_j - y_i \right)^2$$

yielding

$$E' = \frac{dE}{d\mathbf{p}} = \sum_{i=0}^{m-1} \left(2 \cdot \left(\sum_{j=0}^{n-1} x_{i,j} \cdot p_j - y_i \right) \cdot \mathbf{x}_i \right) = 2 \cdot \sum_{i=0}^{m-1} (\mathbf{x}_i \cdot \mathbf{p} - y_i) \cdot \mathbf{x}_i$$

and

$$E'' = \frac{d^2 E}{d\mathbf{p}^2} = \frac{dE'^T}{d\mathbf{p}} = \frac{d\left(2 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot (\mathbf{x}_i \cdot \mathbf{p} - y_i)\right)}{d\mathbf{p}} = 2 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot \mathbf{x}_i \ .$$

Linear Regression

Differentiation


```cpp
template<typename T, int M, int N>
T E(const Eigen::Matrix<T,N,1> &p, const Eigen::Matrix<T,M,N> &x,
    const Eigen::Matrix<T,M,1> &y) {
  return (x*p-y).squaredNorm();
}

template<typename T, int M, int N>
Eigen::Matrix<T,1,N> dEdp(const Eigen::Matrix<T,N,1> &p,
    const Eigen::Matrix<T,M,N> &x, const Eigen::Matrix<T,M,1> &y) {
  auto m=y.size(), n=p.size();
  Eigen::Matrix<T,1,N> r=Eigen::Matrix<T,1,N>::Zero(1,n);
  for (auto i=0;i<m;i++) r+=(x.row(i).dot(p)-y(i))*x.row(i);
  return 2*r;
}

template<typename T, int M, int N>
Eigen::Matrix<T,N,N> ddEdpp(const Eigen::Matrix<T,N,1> &p,
    const Eigen::Matrix<T,M,N> &x, const Eigen::Matrix<T,M,1> &y) {
  auto m=y.size(), n=p.size();
  Eigen::Matrix<T,N,N> r=Eigen::Matrix<T,N,N>::Zero(n,n);
  for (auto i=0;i<m;i++) r+=x.row(i).transpose()*x.row(i);
  return 2*r;
}
```

Linear Regression

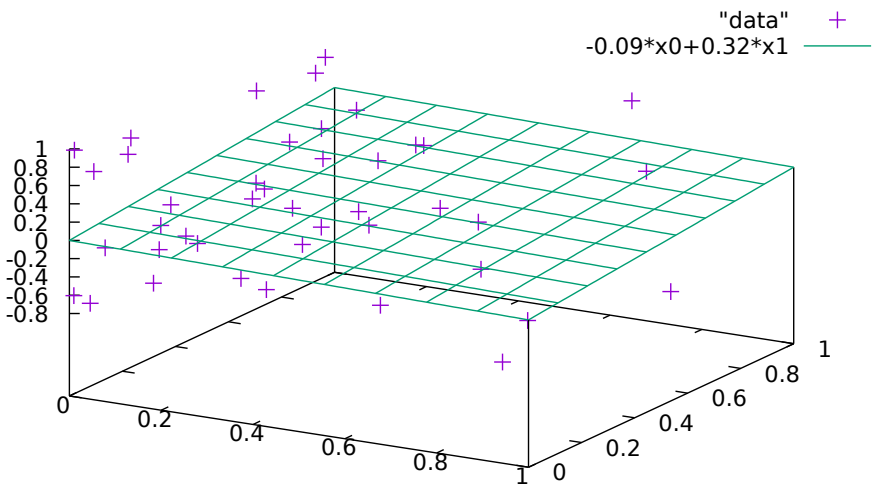
main

```
1  int main(int argc, char* argv[]) {
2    assert(argc==3); auto m=std::stoi(argv[1]), n=std::stoi(argv[2]);
3    using T=double;
4    using MT=Eigen::Matrix<T,Eigen::Dynamic,Eigen::Dynamic>;
5    using VT=Eigen::Matrix<T,Eigen::Dynamic,1>;
6    MT x=MT::Random(m,n); x=x.cwiseProduct(x);
7    VT p(n), y=VT::Random(m);
8    Regression(p,x,y);
9    std::cout << "p^T=" << p.transpose() << std::endl;
10   std::cout << "E'=" << dEdp(p,x,y) << std::endl;
11   std::cout << "E''=" << std::endl << ddEdpp(p,x,y) << std::endl;
12   Eigen::LLT<Eigen::Matrix<T,Eigen::Dynamic,Eigen::Dynamic>> spd(ddEdpp(p,x,y));
13   if (spd.info()!=Eigen::NumericalIssue)
14     std::cout << "Hessian is spd." << std::endl;
15   else
16     std::cout << "Hessian appears to be not spd!" << std::endl;
17   return 0;
18 }
```

Inspection of source code and live demo.

Similar results for Gauss-Newton and orthogonalization methods.

```
1 ./main.exe 42 2
2 p^T=-0.085952 0.315279
3 E'^T=-2.77556e-16 -1.33227e
      -15
4 E''=
5 14.4774 9.34914
6 9.34914 18.8998
7 Hessian is spd.
```



Outline

For given data $(X = (\mathbf{x}_i) = (x_{i,j}), \mathbf{y}) \in \boldsymbol{R}^{m \times n} \times \boldsymbol{R}^m$ to be fitted by a nonlinear model $y = f(\mathbf{p}, \mathbf{x}) = (\mathbf{x}^T \cdot \mathbf{p})^2$ we minimize the error function

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

requiring

$$E' = \frac{dE}{d\mathbf{p}} = 4 \cdot \sum_{i=0}^{m-1} ((\mathbf{x}_i \cdot \mathbf{p})^2 - y_i) \cdot \mathbf{x}_i \cdot \mathbf{p} \cdot \mathbf{x}_i = 0$$

and a positive definite Hessian

$$E'' = \frac{d^2 E}{d\mathbf{p}^2} = 4 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot \mathbf{x}_i \cdot (3 \cdot (\mathbf{x}_i \cdot \mathbf{p})^2 - y_i) \, .$$

```
1 template<typename T, int M, int N>
2 T E(const Eigen::Matrix<T,N,1> &p, const Eigen::Matrix<T,M,N> &x,
3       const Eigen::Matrix<T,M,1> &y) {
4   T o=0;
5   for (auto i=0;i<y.size();i++) o+=pow(pow(x.row(i)*p,2)-y(i),2);
6   return o;
7 }
```

Solution of

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

for

$$F(\mathbf{p}, X, \mathbf{y}) = \left((\mathbf{x}_i \cdot \mathbf{p})^2 - y_i\right)_{i=0}^{m-1}$$

requires

$$F'(\mathbf{p}, X, \mathbf{y}) \equiv \frac{dF}{\mathbf{p}}(\mathbf{p}, X, \mathbf{y}) = 2 \cdot (\mathbf{x}_i \cdot \mathbf{p} \cdot \mathbf{x}_i)_{i=0}^{m-1} \in R^{m \times n} \ .$$

```cpp
template<typename T, int M, int N>
Eigen::Matrix<T,M,1> F(const Eigen::Matrix<T,N,1> &p,
    const Eigen::Matrix<T,M,N> &x, const Eigen::Matrix<T,M,1> &y) {
  int m=y.size();
  Eigen::Matrix<T,M,1> r(m);
  for (auto i=0;i<m;i++)
    r(i)=pow(x.row(i)*p,2)-y(i);
  return r;
}

template<typename T, int M, int N>
Eigen::Matrix<T,M,N> dFdp(const Eigen::Matrix<T,N,1> &p,
    const Eigen::Matrix<T,M,N> &x, const Eigen::Matrix<T,M,1> &y) {
  int m=y.size(), n=p.size();
  Eigen::Matrix<T,M,N> rj(m,n);
  for (auto i=0;i<m;i++) rj.row(i)=x.row(i)*p*x.row(i);
  return 2*rj;
}
```

Differentiation of the error function

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

wrt. to $\mathbf{p}$ yields $E' \in \boldsymbol{R}^{1 \times n}$ as

$$E' = \frac{dE}{d\mathbf{p}} = \frac{d\left(\sum_{i=0}^{m-1} \left((\mathbf{x}_i \cdot \mathbf{p})^2 - y_i \right)^2 \right)}{d\mathbf{p}} = 4 \cdot \sum_{i=0}^{m-1} \left((\mathbf{x}_i \cdot \mathbf{p})^2 - y_i \right) \cdot \mathbf{x}_i \cdot \mathbf{p} \cdot \mathbf{x}_i$$

The Hessian is obtained by differentiation of ${E'}^T$ wrt. $\mathbf{p}$ as

$$E'' = \frac{d^2 E}{d\mathbf{p}^2} = \frac{d{E'}^T}{d\mathbf{p}} \in \boldsymbol{R}^{n \times n} .$$

$$\frac{dE'^T}{d\mathbf{p}} = \frac{d\left(4 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot \mathbf{p}^T \cdot \mathbf{x}_i^T \cdot \left((\mathbf{x}_i \cdot \mathbf{p})^2 - y_i\right)\right)}{d\mathbf{p}}$$
$$= \frac{d\left(4 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot \mathbf{x}_i \cdot \left[\mathbf{p} \cdot \left((\mathbf{x}_i \cdot \mathbf{p})^2 - y_i\right)\right]\right)}{d\mathbf{p}}$$
$$= 4 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot \mathbf{x}_i \cdot \left[(\mathbf{x}_i \cdot \mathbf{p})^2 - y_i + \mathbf{p} \cdot 2 \cdot \mathbf{x}_i \cdot \mathbf{p} \cdot \mathbf{x}_i\right]$$

With

$$\mathbf{x}_i^T \cdot \mathbf{x}_i \cdot \mathbf{p} \cdot \mathbf{x}_i \cdot \mathbf{p} \cdot \mathbf{x}_i = \mathbf{x}_i^T \cdot (\mathbf{x}_i \cdot \mathbf{p})^2 \cdot \mathbf{x}_i = \mathbf{x}_i^T \cdot \mathbf{x}_i \cdot (\mathbf{x}_i \cdot \mathbf{p})^2$$

follows

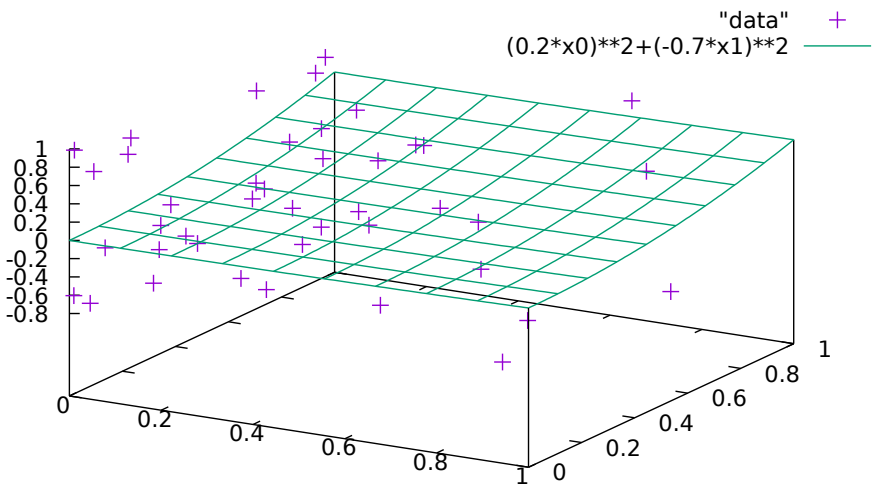
$$E'' = 4 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot \mathbf{x}_i \cdot ((\mathbf{x}_i \cdot \mathbf{p})^2 - y_i + 2 \cdot (\mathbf{x}_i \cdot \mathbf{p})^2) = 4 \cdot \sum_{i=0}^{m-1} \mathbf{x}_i^T \cdot \mathbf{x}_i \cdot (3 \cdot (\mathbf{x}_i \cdot \mathbf{p})^2 - y_i) \ .$$

```
1  template<typename T, int M, int N>
2  Eigen::Matrix<T,1,N> dEdp(const Eigen::Matrix<T,N,1> &p,
3      const Eigen::Matrix<T,M,N> &x, const Eigen::Matrix<T,M,1> &y) {
4    int m=y.size(), n=p.size();
5    Eigen::Matrix<T,1,N> r=Eigen::Matrix<T,1,N>::Zero(1,n);
6    for (auto i=0;i<m;i++)
7      r+=(pow(x.row(i)*p,2)-y(i))*x.row(i)*p*x.row(i);
8    return 4*r;
9  }
10
11 template<typename T, int M, int N>
12 Eigen::Matrix<T,N,N> ddEdpp(const Eigen::Matrix<T,N,1> &p,
13     const Eigen::Matrix<T,M,N> &x, const Eigen::Matrix<T,M,1> &y) {
14   int m=y.size(), n=p.size();
15   Eigen::Matrix<T,N,N> r=Eigen::Matrix<T,N,N>::Zero(n,n);
16   for (auto i=0;i<m;i++)
17     r+=x.row(i).transpose()*x.row(i)*(3*pow(x.row(i)*p,2)-y(i));
18   return 4*r;
19 }
```

Inspection of source code and live demo.

Similar results for Gauss-Newton and orthogonalization methods.

```
1 ./main.exe 42 2
2 p^T=0.196401 -0.69668
3 E'=9.38255e-06 2.64487e-06
4 E''=
5 6.54852 6.15032
6 6.15032 15.5514
7 Hessian is spd.
```



Outline

Summary

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

Next Steps

- Play with the sample code.
- Prepare for final exam