Nonlinear Regression II

Multivariate Scalar Models

Uwe Naumann

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

RWTH Aachen University

Contents

Outline

Objective

- Introduction to nonlinear regression methods for multivariate scalar models.

Learning Outcomes

- You will understand
  - normal equation (Gauss-Newton / Levenberg-Marquardt methods)
  - Givens rotation
  - Householder reflection

  in the context of linearization of nonlinear regression problems.
- You will be able to
  - implement nonlinear regression methods
  - run computational experiments.

Outline

The calibration of nonlinear (in $\mathbf{p} \in R^n$) models to given data $(X, \mathbf{y}) \in R^{m \times n} \times R^m$, $X = (\mathbf{x}_i)_{i=0}^{m-1}$, $m \geq n$, can be posed as a nonlinear minimization problem with objective

$$E(p) = \|F(\mathbf{p}, X, \mathbf{y})\|_2^2 = \sum_{i=0}^{m-1} F_i(\mathbf{p}, X, \mathbf{y})^2 = \sum_{i=0}^{m-1} (f(\mathbf{p}, \mathbf{x}_i) - y_i)^2 \ .$$

Potential solution methods include steepest gradient descent as well as Newton's method applied to the first-order optimality criterion $\frac{dE(\mathbf{p})}{d\mathbf{p}} = 0$ while satisfying the second-order optimality criterion ($\frac{d^2E(\mathbf{p})}{d\mathbf{p}^2}$ symmetric positive definite).

These general-purpose algorithms approximate the solution iteratively up to a given accuracy. Such iteration can be avoided in the linear (in $\mathbf{p}$) case yielding a lower computational complexity of linear regression methods.

Linearization allows for application of these ideas to the nonlinear case.

Formulation of the first-order optimality condition

$$E'(\mathbf{p}) \equiv \frac{dE(\mathbf{p})}{d\mathbf{p}} = \frac{d\|F(\mathbf{p},X,\mathbf{y})\|_2^2}{d\mathbf{p}} = 0$$

in terms of a linearization (linearity in $\Delta\mathbf{p}$) of the residual $F(\mathbf{p},X,\mathbf{y})$ as

$$\frac{d\|F(\mathbf{p},X,\mathbf{y}) + \frac{dF(\mathbf{p},X,\mathbf{y})}{d\mathbf{p}} \cdot \Delta\mathbf{p}\|_2^2}{d\Delta\mathbf{p}} = \frac{d\|F(\mathbf{p},X,\mathbf{y}) + F'(\mathbf{p},X,\mathbf{y}) \cdot \Delta\mathbf{p}\|_2^2}{d\Delta\mathbf{p}} = 0$$

yields a [damped] iterative optimization scheme for $\mathbf{p}$ as

$$\mathbf{p} := \mathbf{p} + [\alpha\cdot]\Delta\mathbf{p}$$

for $0 < \alpha \le 1$. The Jacobian $F'(\mathbf{p},X,\mathbf{y}) \in R^{m\times n}$ is required in addition to $F(\mathbf{p},X,\mathbf{y}) \in R^m$. It can be computed symbolically as well as by finite difference approximation or by algorithmic differentiation.

The linear regression problem

$$F'(\mathbf{p},X,\mathbf{y})\cdot\Delta\mathbf{p}\approx -F(\mathbf{p},X,\mathbf{y})\;,\quad F\in \boldsymbol{R}^m,\; F'\in \boldsymbol{R}^{m\times n},\; \Delta\mathbf{p}\in \boldsymbol{R}^n$$

can be solved with

- the normal equations method
- Givens rotation
- Householder reflection.

Convergence of the fixed-point iteration $\mathbf{p}=G(\mathbf{p})=\mathbf{p}+\Delta\mathbf{p}$ requires

$$\|G'(\mathbf{p})\|<1$$

at the solution $\mathbf{p}^*$ implying existence of a neighborhood of $\mathbf{p}^*$ containing values of $\mathbf{p}$ for which the fixed-point iteration converges to this solution.

Line search potentially improves robustness by extending this neighborhood.

Outline

The linear regression problem

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

yields the normal equation

$$F'(\mathbf{p}, X, \mathbf{y})^T \cdot F'(\mathbf{p}, X, \mathbf{y}) \cdot \Delta\mathbf{p} = -F'(\mathbf{p}, X, \mathbf{y})^T \cdot F(\mathbf{p}, X, \mathbf{y})$$

which can be solved by LL^T (LDL^T) factorization of the symmetric matrix $F'(\mathbf{p}, X, \mathbf{y})^T \cdot F'(\mathbf{p}, X, \mathbf{y}) \in \mathbb{R}^{n \times n}$.

This method is also known as the Gauss-Newton method.

Outlook: Regularization aims for improved numerical stability through reducing singularity of the system matrix and yielding, for example, the Levenberg-Marquardt method

$$\left(F'(\mathbf{p}, X, \mathbf{y})^T \cdot F'(\mathbf{p}, X, \mathbf{y}) - \lambda \cdot I_n\right) \cdot \Delta\mathbf{p} = -F'(\mathbf{p}, X, \mathbf{y})^T \cdot F(\mathbf{p}, X, \mathbf{y}) \; .$$

With $A \equiv F'(\mathbf{p}, X, \mathbf{y}) = (\mathbf{a}_i)_{i=0}^{m-1}$ and $\mathbf{b} = -F(\mathbf{p}, X, \mathbf{y})$ we get

$$\boldsymbol{R}^n \ni 0 = \left(\frac{d\|A \cdot \Delta\mathbf{p} - \mathbf{b}\|_2^2}{d\Delta\mathbf{p}}\right)^T = \left(\frac{d\left[\sum_{i=0}^{m-1}(\mathbf{a}_i \cdot \Delta\mathbf{p} - b_i)^2\right]}{d\Delta\mathbf{p}}\right)^T$$

$$= 2 \cdot \left(\sum_{i=0}^{m-1}(\mathbf{a}_i \cdot \Delta\mathbf{p} - b_i) \cdot \mathbf{a}_i\right)^T = 2 \cdot \sum_{i=0}^{m-1} \mathbf{a}_i^T \cdot (\mathbf{a}_i \cdot \Delta\mathbf{p} - b_i)$$

$$= \sum_{i=0}^{m-1} \mathbf{a}_i^T \cdot \mathbf{a}_i \cdot \Delta\mathbf{p} - \sum_{i=0}^{m-1} \mathbf{a}_i^T \cdot b_i = A^T \cdot A \cdot \Delta\mathbf{p} - A^T \cdot \mathbf{b}$$

implying $A^T \cdot A \cdot \Delta\mathbf{p} = A^T \cdot \mathbf{b}$ and hence

$$F'(\mathbf{p}, X, \mathbf{y})^T \cdot F'(\mathbf{p}, X, \mathbf{y}) \cdot \Delta\mathbf{p} = -F'(\mathbf{p}, X, \mathbf{y})^T \cdot F(\mathbf{p}, X, \mathbf{y})$$

Normal Equation

Implementation

```
template<typename T, int M, int N>
void NormalEquation(const Eigen::Matrix<T,M,N> &A, Eigen::Matrix<T,N,1> &p,
    const Eigen::Matrix<T,M,1> &y) {
  p=(A.transpose()*A).llt().solve(-A.transpose()*y);
}
```

```
template<typename T, int M, int N>
void Regression(Eigen::Matrix<T,N,1> &p, const Eigen::Matrix<T,M,N> &x,
    const Eigen::Matrix<T,M,1> &y, const T& eps) {
  T o=E(p,x,y),o_prev;
  while (dEdp(p,x,y).norm()>eps) {
    o_prev=o;
    Eigen::Matrix<T,N,1> delta_p;
    Eigen::Matrix<T,M,1> r=F(p,x,y);
    Eigen::Matrix<T,M,N> drdp=dFdp(p,x,y);
    NormalEquation(drdp,delta_p,r);
    T alpha=2.;
    while (o_prev<=o) { // line search
      Eigen::Matrix<T,N,1> p_trial=p;
      alpha/=2;
      p_trial+=alpha*delta_p; o=E(p_trial,x,y);
    }
    p+=alpha*delta_p;
  }
}
```

Outline

Orthogonalization of

$$F' \cdot \Delta\mathbf{p} \approx -F \; .$$

yields

$$Q \cdot R \cdot \Delta\mathbf{p} \approx -F$$

with orthogonal $Q \in \boldsymbol{R}^{m \times m}$ and upper triangular $R \in \boldsymbol{R}^{m \times n}$.

The resulting system of linear equations

$$[R]_{0,\ldots,n-1} \cdot \Delta\mathbf{p} = \left[-Q^T \cdot F\right]_{0,\ldots,n-1}$$

is easily solved by backward substitution and followed by updating **p** as

$$\mathbf{p} := \mathbf{p} + \alpha \cdot \Delta\mathbf{p}$$

starting from a suitable initial estimate to ensure convergence.

Solution of the linear regression problem

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

by Givens rotation transforms the matrix

$$F'(\mathbf{p}, X, \mathbf{y}) \in \boldsymbol{R}^{m \times n} \quad \text{into the upper triangular} \quad R = Q^T \cdot F'(\mathbf{p}, X, \mathbf{y})$$

followed by the solution of

$$Q^T \cdot F'(\mathbf{p}, X, \mathbf{y}) \cdot \Delta\mathbf{p} \approx -Q^T \cdot F(\mathbf{p}, X, \mathbf{y})$$

yielding $\Delta\mathbf{p}$ as the solution of the

$$[R]_{0,\ldots,n-1} \cdot \Delta\mathbf{p} = \left[-Q^T \cdot F\right]_{0,\ldots,n-1} \ .$$

See module LinearRegression_II for derivation.

Givens Rotation

Implementation

```cpp
template<typename T, int M, int N>
void Givens(const Eigen::Matrix<T,M,N> &A, Eigen::Matrix<T,M,M> &QT,
    Eigen::Matrix<T,M,N> &R) {
  int m=A.rows(), n=A.cols();
  using MT=Eigen::Matrix<T,M,M>;
  R=A; QT=MT::Identity(m,m);
  for (int j=0;j<n;j++) {
    for (int i=m-2;i>=j;i--) {
      T norm_a_tilde=R.col(j).block(i,0,2,1).norm();
      MT G=MT::Identity(m,m);
      G(i,i)=R.col(j)(i)/norm_a_tilde; G(i+1,i+1)=G(i,i);
      G(i,i+1)=R.col(j)(i+1)/norm_a_tilde; G(i+1,i)=-G(i,i+1);
      QT=G*QT; R=G*R;
    }
  }
}
```

```
template<typename T, int M, int N>
void LinearSolve(const Eigen::Matrix<T,M,M> &QT, const Eigen::Matrix<T,M,N> &R,
    Eigen::Matrix<T,N,1> &p, const Eigen::Matrix<T,M,1> &y) {
  int n=R.cols();
  Eigen::Matrix<T,M,1> r=QT*y;
  for (int i=n-1;i>=0;i--) {
    T d=r(i);
    for (int j=n-1;j>i;j--) d-=R(i,j)*p(j);
    p(i)=d/R(i,i);
  }
}
```

Givens Rotation

Implementation

```cpp
template<typename T, int M, int N>
void Regression(Eigen::Matrix<T,N,1> &p, const Eigen::Matrix<T,M,N> &x,
    const Eigen::Matrix<T,M,1> &y, const T& eps) {
  int m=y.size(), n=p.size();
  T o=E(p,x,y),o_prev;
  while (dEdp(p,x,y).norm()>eps) {
    o_prev=o;
    Eigen::Matrix<T,N,1> delta_p(n);
    Eigen::Matrix<T,M,1> r=F(p,x,y);
    Eigen::Matrix<T,M,N> drdp=dFdp(p,x,y);
    Eigen::Matrix<T,M,M> QT(m,m); Eigen::Matrix<T,M,N> R(m,n);
    Givens(drdp,QT,R);
    LinearSolve(QT,R,delta_p,r);
    T alpha=2.;
    while (o_prev<=o) {
      Eigen::Matrix<T,N,1> p_trial=p;
      alpha/=2;
      p_trial-=alpha*delta_p; o=E(p_trial,x,y);
    }
    p-=alpha*delta_p;
  }
}
```

Solution of the linear regression problem

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

by Householder reflection transforms the matrix

$$F'(\mathbf{p}, X, \mathbf{y}) \in \boldsymbol{R}^{m \times n} \quad \text{into to upper triangular} \quad R = Q^T \cdot F'(\mathbf{p}, X, \mathbf{y})$$

followed by the solution of

$$Q^T \cdot F'(\mathbf{p}, X, \mathbf{y}) \cdot \Delta\mathbf{p} \approx -Q^T \cdot F(\mathbf{p}, X, \mathbf{y})$$

yielding $\Delta\mathbf{p}$ as the solution of the

$$[R]_{0,\ldots,n-1} \cdot \Delta\mathbf{p} = \left[-Q^T \cdot F\right]_{0,\ldots,n-1} \ .$$

See module LinearRegression_II for derivation.

```
template<typename T, int M, int N>
void Householder(const Eigen::Matrix<T,M,N> &A, Eigen::Matrix<T,M,M> &QT,
    Eigen::Matrix<T,M,N> &R) {
  int m=A.rows(), n=A.cols();
  using VT=Eigen::Matrix<T,M,1>;
  using MT=Eigen::Matrix<T,M,M>;
  R=A; QT=MT::Identity(m,m);
  for (int j=0;j<n;j++) {
    VT x=R.col(j);
    for (int i=0;i<j;i++) x(i)=0;
    VT v=x+x(j)/fabs(x(j))*x.norm()*VT::Unit(m,j);
    MT H=MT::Identity(m,m)-2*v*v.transpose()/v.dot(v);
    QT=H*QT; R=H*R;
  }
}
```

Householder Reflection

Implementation

```
template<typename T, int M, int N>
void LinearSolve(const Eigen::Matrix<T,M,M> &QT, const Eigen::Matrix<T,M,N> &R,
    Eigen::Matrix<T,N,1> &p, const Eigen::Matrix<T,M,1> &y) {
  int n=R.cols();
  Eigen::Matrix<T,M,1> r=QT*y;
  for (int i=n-1;i>=0;i--) {
    T d=r(i);
    for (int j=n-1;j>i;j--) d-=R(i,j)*p(j);
    p(i)=d/R(i,i);
  }
}
```

Householder Reflection

Implementation

```
template<typename T, int M, int N>
void Regression(Eigen::Matrix<T,N,1> &p, const Eigen::Matrix<T,M,N> &x,
    const Eigen::Matrix<T,M,1> &y, const T& eps) {
  int m=y.size(), n=p.size();
  T o=E(p,x,y),o_prev;
  while (dEdp(p,x,y).norm()>eps) {
    o_prev=o;
    Eigen::Matrix<T,N,1> delta_p(n);
    Eigen::Matrix<T,M,1> r=F(p,x,y);
    Eigen::Matrix<T,M,N> drdp=dFdp(p,x,y);
    Eigen::Matrix<T,M,M> QT(m,m); Eigen::Matrix<T,M,N> R(m,n);
    Householder(drdp,QT,R);
    LinearSolve(QT,R,delta_p,r);
    T alpha=2.;
    while (o_prev<=o) { // line search
      Eigen::Matrix<T,N,1> p_trial=p;
      alpha/=2;
      p_trial-=alpha*delta_p; o=E(p_trial,x,y);
    }
    p-=alpha*delta_p;
  }
}
```

Outline

Summary

- Nonlinear regression methods for multivariate scalar models based on linearization and
  - normal equation (Gauss-Newton / Levenberg-Marquardt methods)
  - Givens rotation
  - Householder reflection

Next Steps

- Play with sample code.
- Compare results with those obtained by convex minimization methods.
- Continue the course to find out more ...