Newton's Method II

Roots and Stationary Points of Multivariate Functions

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering

RWTH Aachen University

Outline

Outline

Objective

- Introduction to Newton's method for multivariate functions.

Learning Outcomes

- You will understand
  - systems of nonlinear equations
  - linearization
  - convergence
  - stationary points and local optima
- You will be able to
  - implement Steepest Descent method
  - implement Newton's method
  - investigate convergence of Newton's method.

Outline

Similar to the scalar (1D) case, many numerical methods for nonlinear problems in nD are built on local replacement of the target function with a linear (affine) approximation derived from the truncated Taylor series expansion and "hoping" that

$$F(\mathbf{x}+\Delta\mathbf{x}) \approx F(\mathbf{x}) + F'(\mathbf{x}) \cdot \Delta\mathbf{x}$$

i.e, relying on the assumption that the remainder is reasonably small within the subdomain of interest.

The solution of a sequence of linear problems is expected to yield an iterative approximation of the solution to the nonlinear problem.

THE example is the solution of systems of nonlinear equations as a sequence of solutions of linear problems known as Newton's method.

When consider a system of nonlinear equations $\mathbf{y} = F(\mathbf{x}) = 0$ under the assumption that

$$\bar{F}(\mathbf{x}, \Delta\mathbf{x}) \equiv F(\mathbf{x}) + F'(\mathbf{x}) \cdot \Delta\mathbf{x} \approx F(\mathbf{x} + \Delta\mathbf{x})$$

at some point $\mathbf{x} \in \boldsymbol{R}^n$ the root finding problem for F can be replaced locally by the root finding problem for the linearization $\bar{F}(\mathbf{x}, \Delta\mathbf{x})$.

Solution of the system of linear (in $\Delta\mathbf{x}$) equations $\bar{F}(\mathbf{x}, \Delta\mathbf{x}) = 0$ yields

$$\Delta\mathbf{x} = -F'(\mathbf{x})^{-1} \cdot F(\mathbf{x})$$

such that $F(\mathbf{x} + \Delta\mathbf{x}) \approx 0$. The Jacobian $F'(\mathbf{x})$ needs to be invertible. The Newton step $\Delta\mathbf{x}$ is computed as the solution to the linear system

$$F'(\mathbf{x}) \cdot \Delta\mathbf{x} = -F(\mathbf{x})$$

If the new iterate is not close enough to the root, i.e, $\|F(\mathbf{x}+\Delta\mathbf{x})\| > \epsilon$ for some measure of accuracy of the numerical approximation $1 \gg \epsilon > 0$, then it becomes the starting point for the next iteration yielding the recurrence

$$\begin{aligned} F'(\mathbf{x}) \cdot \Delta\mathbf{x} &= -F(\mathbf{x}) \\ \mathbf{x} &:= \mathbf{x} + \Delta\mathbf{x} \end{aligned}$$

Convergence of Newton's method is not guaranteed in general; see also below. Damping of the magnitude of the next step may help.

$$\mathbf{x} := \mathbf{x} - \alpha \cdot F'(\mathbf{x})^{-1} \cdot F(\mathbf{x}) \quad \text{for } 0 < \alpha \leq 1 \, .$$

The damping parameter α can be determined by line search (e.g, recursive bisection yielding $\alpha = 1, \ 0.5, \ 0.25, \ \ldots$) such that decrease in absolute function value is ensured.

```
#include "Eigen/Dense"

template<typename T, int N=Eigen::Dynamic>
void F(const Eigen::Matrix<T,N,1>& x, Eigen::Matrix<T,N,1>& y);

template<typename T, int N=Eigen::Dynamic>
void dFdx(const Eigen::Matrix<T,N,1>& x, Eigen::Matrix<T,N,N>& dydx);

template<typename T, int N=Eigen::Dynamic>
void Newton(Eigen::Matrix<T,N,1>& x, const T eps) {
  auto n=x.size();
  Eigen::Matrix<T,N,1> y(n);
  F(x,y);
  do {
    Eigen::Matrix<T,N,N> dydx(n,n);
    dFdx(x,dydx);
    x+=dydx.partialPivLu().solve(-y);
    F(x,y);
  } while (y.norm()>eps);
}
```

Newton's method can be regarded as a fixed-point iteration

$$\mathbf{x} = G(\mathbf{x}) \ .$$

If at the solution

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

then there exists a neighborhood containing values of $\mathbf{x}$ for which the fixed-point iteration converges to this solution.

The convergence rate of a fixed-point iteration grows linearly with decreasing values of $\|G'(\mathbf{x})\|$.

For $\|G'(\mathbf{x})\| = 0$ we get at least quadratic convergence; cubic for $\|G'(\mathbf{x})\| = \|G''(\mathbf{x})\| = 0$ and so forth.

Newton's method becomes

$$\mathbf{x} = G(\mathbf{x}) = \mathbf{x} - F'(\mathbf{x})^{-1} \cdot F(\mathbf{x}) = \mathbf{x} - \Delta\mathbf{x}$$

yielding

$$G'(\mathbf{x}) = \underbrace{I_n - F'(\mathbf{x})^{-1} \cdot F'(\mathbf{x})}_{=0} - \underbrace{\underbrace{F'(\mathbf{x})^{-1}}_{n \text{ LS}} \cdot F''(\mathbf{x}) \cdot \underbrace{F'(\mathbf{x})^{-1} \cdot F(\mathbf{x})}_{\text{LS}}}_{\text{second-order tangent}}$$

At the solution $F(\mathbf{x}) = 0$ implies $G'(\mathbf{x}) = 0$. Assuming a simple root ($F(\mathbf{x}) = 0$, $F'(\mathbf{x}) \neq 0$) the second derivative of G becomes equal to

$$G''(\mathbf{x}) = F'(\mathbf{x})^{-1} \cdot F''(\mathbf{x}) \cdot \underbrace{F'(\mathbf{x})^{-1} \cdot F'(\mathbf{x})}_{=I_n} + (\ldots) \cdot \underbrace{F(\mathbf{x})}_{=0}$$

implying quadratic convergence within the corresponding neighborhood of the solution if $F''(\mathbf{x}) \neq 0$ as well as convergence after a single iteration for linear F.

Outline

Let $y = f(\mathbf{x}) : R^n \to R$ be twice continuously differentiable.

- $\tilde{\mathbf{x}}$ is a stationary point of f if

$$f'(\tilde{\mathbf{x}}) \equiv \frac{df}{d\mathbf{x}}(\tilde{\mathbf{x}}) = 0 \; .$$

- $\tilde{\mathbf{x}}$ is a local minimum of f if the Hessian $f''(\tilde{\mathbf{x}}) \equiv \frac{d^2 f}{d\mathbf{x}^2}(\tilde{\mathbf{x}})$ is symmetric positive definite, i.e,

$$\forall \mathbf{v} \neq 0 \in R^n : \quad \mathbf{v}^T \cdot f''(\tilde{\mathbf{x}}) \cdot \mathbf{v} > 0 \quad \text{(strict convexity)}.$$

- $\tilde{\mathbf{x}}$ is a local maximum of f if the Hessian is symmetric negative definite, i.e,

$$\forall \mathbf{v} \neq 0 \in R^n : \quad \mathbf{v}^T \cdot f''(\tilde{\mathbf{x}}) \cdot \mathbf{v} < 0 \quad \text{(strict concavity)}.$$

- $f''(\tilde{\mathbf{x}}) = 0$ indicates a non-simple stationary point.

We consider the unconstrained nonlinear optimization problem $\min_{\mathbf{x}\in R^n} f(\mathbf{x})$.

Starting from some initial estimate for $\tilde{\mathbf{x}}$ steps into descent directions are taken.

The gradient f' of f wrt. $\mathbf{x}$ indicates local increase ($\|f'\| > 0$) or decrease ($\|f'\| < 0$) of the function value.

Aiming for decrease the next step should be in direction of the negative gradient $-f'$. No further local decrease in the function value can be achieved for $\|f'\| = 0$ (necessary optimality condition).

The step size is typically damped in order to ensure continued progress toward the minimum yielding the recurrence

$$\mathbf{x} := \mathbf{x} - \alpha \cdot f'(\mathbf{x}) \quad \text{while } \|f'(\mathbf{x})\| > \epsilon \,.$$

The damping parameter is often determined by line search (e.g, recursive bisection yielding $\alpha = 1,\ 0.5,\ 0.25,\ \ldots$) such that decrease in function value is ensured.

```
#include "Eigen/Dense"

template<typename T, int N=Eigen::Dynamic>
void f(const Eigen::Matrix<T,N,1>& x, T& y);

template<typename T, int N=Eigen::Dynamic>
void dfdx(const Eigen::Matrix<T,N,1>& x, Eigen::Matrix<T,N,1>& dydx);

template<typename T, int N=Eigen::Dynamic>
void SteepestDescent(Eigen::Matrix<T,N,1>& x, const T eps) {
  auto n=x.size();
  Eigen::Matrix<T,N,1> dydx(n); T y, y_prev;
  f(x,y); dfdx(x,dydx);
  do {
    y_prev=y; double alpha=2.;
    while (y_prev<=y) {
      Eigen::Matrix<T,N,1> x_trial=x; alpha/=2;
      x_trial-=alpha*dydx; f(x_trial,y);
    }
    x-=alpha*dydx; dfdx(x,dydx);
  } while (dydx.norm()>eps);
}
```

We consider the unconstrained nonlinear optimization problem $\min_{\mathbf{x}\in R^n} f(\mathbf{x})$.

Linearization of the first-order optimality condition yields a sequence (for evolving $\mathbf{x}$) of local linear (in $\Delta\mathbf{x} \in R^n$) approximations of the first-order optimality condition as

$$f'(\mathbf{x}+\Delta\mathbf{x}) = f'(\mathbf{x}) + f''(\mathbf{x}) \cdot \Delta\mathbf{x} = 0$$

and hence systems of linear equations

$$f'' \cdot \Delta\mathbf{x} = -f'$$

to be solved for $\Delta\mathbf{x}$ and followed by updating

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

iteratively for a suitable start value $\mathbf{x}$ and damping parameter $R \ni \alpha > 0$ determined by line search.

```
template<typename T, int N=Eigen::Dynamic>
void f(const Eigen::Matrix<T,N,1>& x, Eigen::Matrix<T,N,1>& y);

template<typename T, int N=Eigen::Dynamic>
void dfdx(const Eigen::Matrix<T,N,1>& x, Eigen::Matrix<T,N,1>& dydx);

template<typename T, int N=Eigen::Dynamic>
void ddfdxx(const Eigen::Matrix<T,N,1>& x, Eigen::Matrix<T,N,N>& ddydxx);

template<typename T, int N=Eigen::Dynamic>
void Newton(Eigen::Matrix<T,N,1>& x, const T eps) {
  auto n=x.size();
  Eigen::Matrix<T,N,1> dydx(n);
  dfdx(x,dydx);
  do {
    Eigen::Matrix<T,N,N> ddydxx(n,n);
    ddfdxx(x,ddydxx);
    x+=ddydxx.llt().solve(-dydx);
    dfdx(x,dydx);
  } while (dydx.norm()>eps);
}
```

Outline

Summary

- Newton's method for systems of nonlinear equations
- Steepest Descent optimization method
- Newton's method for optimization

Next Steps

- Inspect sample code.
- Run further experiments.
- Continue the course to find out more ...