最佳化方法與應用
MA5037-*

Chapter 11. Nonlinear Equations

Introduction

In many applications we do not need to optimize an objective function explicitly, but rather to find values of the variables in a model that satisfy a number of given relationships. When these relationships take the form of n equalities – the same number of equality conditions as variables in the model – the problem is one of solving a system of nonlinear equations. We write this problem mathematically as

$$r(x) = 0\,, \tag{1}$$

where $r : \mathbb{R}^n \to \mathbb{R}^n$ is a vector-valued function; that is,

$$r(x) = \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_n(x) \end{bmatrix}.$$

Introduction

In this chapter, we assume that each function $r_j : \mathbb{R}^n \to \mathbb{R}$, $j = 1, 2, \cdots, n$, is smooth. A vector x_* for which (1) is satisfied is called a solution or root of the nonlinear equations. In general, the system (1) may have no solutions, a unique solution, or many solutions.

The techniques for solving nonlinear equations overlap in their motivation, analysis, and implementation with optimization techniques discussed in earlier chapters. In both optimization and nonlinear equations, Newton'smethod lies at the heart of many important algorithms. Features such as line searches, trust regions, and inexact solution of the linear algebra sub-problems at each iteration are important in both areas, as are other issues such as derivative evaluation and global convergence.

Introduction

In this chapter, we assume that each function $r_j : \mathbb{R}^n \to \mathbb{R}$, $j = 1, 2, \cdots, n$, is smooth. A vector x_* for which (1) is satisfied is called a solution or root of the nonlinear equations. In general, the system (1) may have no solutions, a unique solution, or many solutions.

The techniques for solving nonlinear equations overlap in their motivation, analysis, and implementation with optimization techniques discussed in earlier chapters. In both optimization and nonlinear equations, Newton'smethod lies at the heart of many important algorithms. Features such as line searches, trust regions, and inexact solution of the linear algebra sub-problems at each iteration are important in both areas, as are other issues such as derivative evaluation and global convergence.

Introduction

Because some important algorithms for nonlinear equations proceed by minimizing a sum of squares of the equations; that is,

$$\min_x \sum_{j=1}^{n} r_j^2(x)\,,$$

there are particularly close connections with the nonlinear least-squares problem discussed in Chapter 10. The differences are that in nonlinear equations, the number of equations equals the number of variables (instead of exceeding the number of variables, as is typically the case in Chapter 10), and that we expect all equations to be satisfied at the solution, rather than just minimizing the sum of squares. This point is important because the nonlinear equations may represent physical or economic constraints such as conservation laws or consistency principles, which must hold exactly in order for the solution to be meaningful.

Introduction

Because some important algorithms for nonlinear equations proceed by minimizing a sum of squares of the equations; that is,

$$\min_x \sum_{j=1}^{n} r_j^2(x)\,,$$

there are particularly close connections with the nonlinear least-squares problem discussed in Chapter 10. The differences are that in nonlinear equations, the number of equations equals the number of variables (instead of exceeding the number of variables, as is typically the case in Chapter 10), and that we expect all equations to be satisfied at the solution, rather than just minimizing the sum of squares. This point is important because the nonlinear equations may represent physical or economic constraints such as conservation laws or consistency principles, which must hold exactly in order for the solution to be meaningful.

Introduction

Many applications require us to solve a sequence of closely related nonlinear systems, as in the following example.

Example

An interesting problem in control is to analyze the stability of an aircraft in response to the commands of the pilot. The following is a simplified model based on force-balance equations, in which gravity terms have been neglected.

Introduction

Example (cont'd)

The equilibrium equations for a particular aircraft are given by a system of 5 equations in 8 unknowns of the form

$$F(x) \equiv Ax + \varphi(x) = 0\,,$$

where $F : \mathbb{R}^8 \to \mathbb{R}^5$, the matrix A is given by

$$A = \begin{bmatrix} -3.933 & 0.107 & 0.126 & 0 & -9.99 & 0 & -45.83 & -7.64 \\ 0 & -0.987 & 0 & -22.95 & 0 & -28.37 & 0 & 0 \\ 0.002 & 0 & -0.235 & 0 & 5.67 & 0 & -0.921 & -6.51 \\ 0 & 1.0 & 0 & -1.0 & 0 & -0.168 & 0 & 0 \\ 0 & 0 & -1.0 & 0 & -0.196 & 0 & -0.0071 & 0 \end{bmatrix},$$

and the nonlinear part is defined by

$$\varphi(x) = \begin{bmatrix} -0.727x_2x_3 + 8.39x_3x_4 - 684.4x_4x_5 + 63.5x_4x_2 \\ 0.949x_1x_3 + 0.173x_1x_5 \\ -0.716x_1x_2 - 1.578x_1x_4 + 1.132x_4x_2 \\ -x_1x_5 \\ x_1x_4 \end{bmatrix}.$$

Introduction

Example (cont'd)

The first three variables x_1, x_2, x_3, represent the rates of roll, pitch, and yaw, respectively, while x_4 is the incremental angle of attack and x_5 the sideslip angle. The last three variables x_6, x_7, x_8 are the controls; they represent the deflections of the elevator, aileron, and rudder, respectively.

For a given choice of the control variables x_6, x_7, x_8 we obtain a system of 5 equations and 5 unknowns. If we wish to study the behavior of the aircraft as the controls are changed, we need to solve a system of nonlinear equations with unknowns x_1, x_2, $\cdots$, x_5 for each setting of the controls.

Introduction

Despite the similarities between nonlinear equations and unconstrained and least-squares optimization algorithms, there are also some important differences. To obtain quadratic convergence in optimization we require second derivatives of the objective function, whereas knowledge of the first derivatives is sufficient in nonlinear equations.

Quasi-Newton methods are perhaps less useful in nonlinear equations than in optimization. In unconstrained optimization, the objective function is the natural choice of merit function that gauges progress towards the solution, but in nonlinear equations various merit functions can be used, all of which have some drawbacks. Line search and trust-region techniques play an equally important role in optimization, but one can argue that trust-region algorithms have certain theoretical advantages in solving nonlinear equations.

Introduction

Despite the similarities between nonlinear equations and unconstrained and least-squares optimization algorithms, there are also some important differences. To obtain quadratic convergence in optimization we require second derivatives of the objective function, whereas knowledge of the first derivatives is sufficient in nonlinear equations.

Quasi-Newton methods are perhaps less useful in nonlinear equations than in optimization. In unconstrained optimization, the objective function is the natural choice of merit function that gauges progress towards the solution, but in nonlinear equations various merit functions can be used, all of which have some drawbacks. Line search and trust-region techniques play an equally important role in optimization, but one can argue that trust-region algorithms have certain theoretical advantages in solving nonlinear equations.

Introduction

Some of the difficulties that arise in trying to solve nonlinear equations can be illustrated by a simple scalar example ($n = 1$). Suppose we have

$$r(x) = \sin(5x) - x, \tag{2}$$

as plotted in Figure 1 (in the next slide). From the figure we see that there are three solutions of the problem $r(x) = 0$, also known as roots of r, located at zero and approximately ± 0.519148. This situation of multiple solutions is similar to optimization problems where, for example, a function may have more than one local minimum. It is not quite the same, however: In the case of optimization, one of the local minima may have a lower function value than the others (making it a "better" solution), while in nonlinear equations all solutions are equally good from a mathematical viewpoint.

Introduction

Some of the difficulties that arise in trying to solve nonlinear equations can be illustrated by a simple scalar example ($n = 1$). Suppose we have

$$r(x) = \sin(5x) - x, \tag{2}$$

as plotted in Figure 1 (in the next slide). From the figure we see that there are three solutions of the problem $r(x) = 0$, also known as roots of r, located at zero and approximately ± 0.519148. This situation of multiple solutions is similar to optimization problems where, for example, a function may have more than one local minimum. It is not quite the same, however: In the case of optimization, one of the local minima may have a lower function value than the others (making it a "better" solution), while in nonlinear equations all solutions are equally good from a mathematical viewpoint.

Introduction

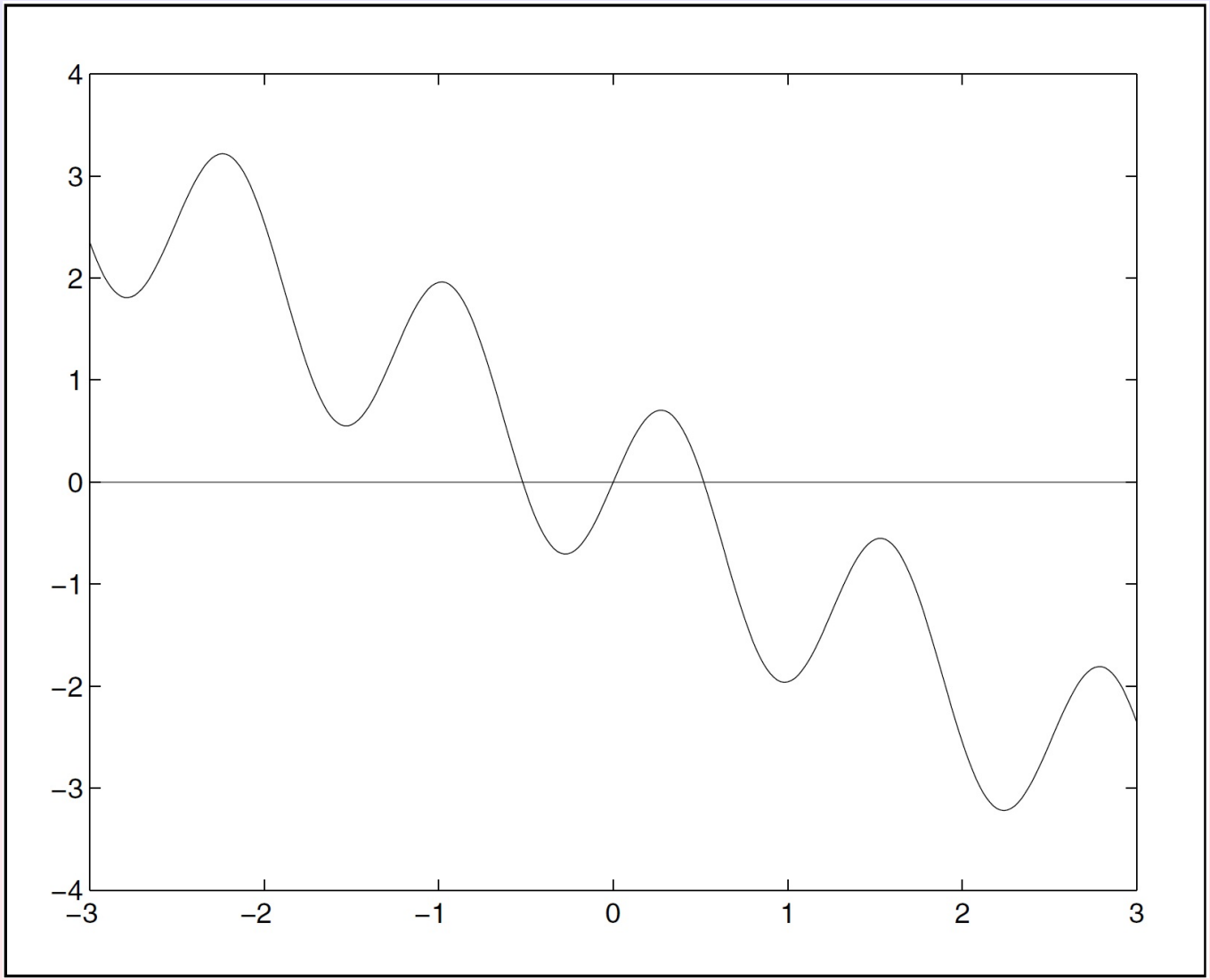

Figure 1: The function $r(x) = \sin(5x) - x$ has three roots.

Introduction

In this chapter we start by outlining algorithms related to Newton's method and examining their local convergence properties. Besides Newton's method itself, these include Broyden's quasi-Newton method, inexact Newton methods, and tensor methods. We then address global convergence, which is the issue of trying to force convergence to a solution from a remote starting point. Finally, we discuss a class of methods in which an "easy" problem – one to which the solution is well known – is gradually transformed into the problem $F(x) = 0$. In these so-called continuation (or homotopy) methods, we track the solution as the problem changes, with the aim of finishing up at a solution of $F(x) = 0$.

Introduction

Throughout this chapter we make the assumption that the vector function r is continuously differentiable in the region D containing the values of x we are interested in. In other words, the Jacobian $J(x)$ exists and is continuous. We say that x_* satisfying $r(x_*) = 0$ is a degenerate solution if $J(x_*)$ is singular, and a non-degenerate solution otherwise.

§11.1 Local Algorithms

- **Newton's method for nonlinear equations**

Recall from Taylor's Theorem that Newton's method for minimizing $f : \mathbb{R}^n \to \mathbb{R}$ forms a quadratic model function by taking the first three terms of the Taylor series approximation of f around the current iterate x_k. The Newton step is the vector that minimizes this model. In the case of nonlinear equations, Newton's method is derived in a similar way, but with a linear model, one that involves function values and first derivatives of the functions $r_j(x)$, $i = 1, 2, \cdots, m$ at the current iterate x_k. We justify this strategy by referring to the following multi-dimensional variant of Taylor's theorem.

§11.1 Local Algorithms

Theorem

Suppose that $r : \mathbb{R}^n \to \mathbb{R}^n$ *is continuously differentiable in some convex open set* $\mathcal{D}$ *and that* x *and* $x+p$ *are vectors in* $\mathcal{D}$. *We then have that*

$$r(x+p) = r(x) + \int_0^1 J(x+tp)p\,dt. \tag{3}$$

We can define a linear model $M_k(p)$ of $r(x_k+p)$ by approximating the second term on the right-hand side of (3) by $J(x)p$, and writing

$$M_k(p) \equiv r(x_k) + J(x_k)p\,. \tag{4}$$

Newton's method, in its pure form, chooses the step p_k to be the vector for which $M_k(p_k) = 0$; that is, $p_k = -J(x_k)^{-1}r(x_k)$. We define it formally as follows.

§11.1 Local Algorithms

Theorem

Suppose that $r : \mathbb{R}^n \to \mathbb{R}^n$ *is continuously differentiable in some convex open set* $\mathcal{D}$ *and that* x *and* $x+p$ *are vectors in* $\mathcal{D}$. *We then have that*

$$r(x+p) = r(x) + \int_0^1 J(x+tp)p\,dt. \tag{3}$$

We can define a linear model $M_k(p)$ of $r(x_k+p)$ by approximating the second term on the right-hand side of (3) by $J(x)p$, and writing

$$M_k(p) \equiv r(x_k) + J(x_k)p\,. \tag{4}$$

Newton's method, in its pure form, chooses the step p_k to be the vector for which $M_k(p_k) = 0$; that is, $p_k = -J(x_k)^{-1}r(x_k)$. We define it formally as follows.

§11.1 Local Algorithms

Algorithm 11.1 (Newton's Method for Nonlinear Equations).

Choose x_0;

for $k = 0, 1, 2, \cdots$

Calculate a solution p_k to the Newton equations

$$J(x_k)p_k = -r(x_k); \tag{5}$$

$x_{k+1} \leftarrow x_k + p_k$;

end (for)

We use a linear model to derive the Newton step, rather than a quadratic model as in unconstrained optimization, because the linear model normally has a solution and yields an algorithm with rapid convergence properties. In fact, Newton's method for unconstrained optimization can be derived by applying Algorithm 11.1 to the nonlinear equations $\nabla f(x) = 0$.

§11.1 Local Algorithms

Algorithm 11.1 (Newton's Method for Nonlinear Equations).

Choose x_0;

for $k = 0, 1, 2, \cdots$

Calculate a solution p_k to the Newton equations

$$J(x_k)p_k = -r(x_k); \tag{5}$$

$x_{k+1} \leftarrow x_k + p_k$;

end (for)

We use a linear model to derive the Newton step, rather than a quadratic model as in unconstrained optimization, because the linear model normally has a solution and yields an algorithm with rapid convergence properties. In fact, Newton's method for unconstrained optimization can be derived by applying Algorithm 11.1 to the nonlinear equations $\nabla f(x) = 0$.

§11.1 Local Algorithms

Another connection is with the Gauss-Newton method for nonlinear least squares; the formula (5) is equivalent to

$$J_k^{\mathrm{T}} J_k p_k^{\mathrm{GN}} = -J_k^{\mathrm{T}} r_k \qquad (23)_{10}$$

in the usual case in which $J_k = J(x_k)$ is non-singular. When the iterate x_k is close to a non-degenerate root x_*, Newton's method converges superlinearly, as we show in the theorem stated later. Potential shortcomings of the method include the following.

1. When x_0 is far from x_*, Algorithm 11.1 can behave erratically. When J_k is singular, the Newton step may not even be defined.
2. First-derivative information may be difficult to obtain.
3. It may be too expensive to find the exact Newton step p_k.
4. The root x_* in question may be degenerate; that is, $J(x_*)$ may be singular.

§11.1 Local Algorithms

Another connection is with the Gauss-Newton method for nonlinear least squares; the formula (5) is equivalent to

$$J_k^{\mathrm{T}} J_k p_k^{\mathrm{GN}} = -J_k^{\mathrm{T}} r_k \qquad (23)_{10}$$

in the usual case in which $J_k = J(x_k)$ is non-singular. When the iterate x_k is close to a non-degenerate root x_*, Newton's method converges superlinearly, as we show in the theorem stated later. Potential shortcomings of the method include the following.

1. When x_0 is far from x_*, Algorithm 11.1 can behave erratically. When J_k is singular, the Newton step may not even be defined.
2. First-derivative information may be difficult to obtain.
3. It may be too expensive to find the exact Newton step p_k.
4. The root x_* in question may be degenerate; that is, $J(x_*)$ may be singular.

§11.1 Local Algorithms

An example of a degenerate problem is the scalar function $r(x) = x^2$, which has a single degenerate root at $x_* = 0$. Algorithm 11.1, when started from any nonzero x_0, generates the sequence of iterates $x_k = \frac{1}{2^k} x_0$, which converges to the solution 0, but only at a linear rate.

As we show later in this chapter, Newton's method can be modified and enhanced in various ways to get around most of these problems. The variants we describe form the basis of much of the available software for solving nonlinear equations.

We summarize the local convergence properties of Algorithm 11.1 in the following theorem.

§11.1 Local Algorithms

Theorem

Suppose that r is continuously differentiable in a convex open set $\mathcal{D} \subseteq \mathbb{R}^n$. *Let* $x_* \in \mathcal{D}$ *be a non-degenerate solution of* $r(x) = 0$, *and let* $\{x_k\}$ *be the sequence of iterates generated by Algorithm 11.1. Then when* $x_k \in \mathcal{D}$ *is sufficiently close to* x_*, *we have*

$$\|x_{k+1} - x_*\| = o(\|x_k - x_*\|), \tag{6}$$

indicating local Q-superlinear convergence. If in addition r is Lipschitz continuously differentiable in a neighborhood $\mathcal{N}$ *of* x_*; *that is,*

$$\|\nabla r(x_0) - \nabla r(x_1)\| \leqslant \beta_L \|x_0 - x_1\| \qquad \forall\, x_0, x_1 \in \mathcal{N}$$

for some $\beta_L > 0$, *we have for all* x_k *sufficiently close to* x_* *that*

$$\|x_{k+1} - x_*\| = \mathcal{O}(\|x_k - x_*\|^2), \tag{7}$$

indicating local Q-quadratic convergence.

§11.1 Local Algorithms

Theorem

Suppose that r is continuously differentiable in a convex open set $\mathcal{D} \subseteq \mathbb{R}^n$. *Let* $x_* \in \mathcal{D}$ *be a non-degenerate solution of* $r(x) = 0$, *and let* $\{x_k\}$ *be the sequence of iterates generated by Algorithm 11.1. Then when* $x_k \in \mathcal{D}$ *is sufficiently close to* x_*, *we have*

$$\|x_{k+1} - x_*\| = o(\|x_k - x_*\|), \tag{6}$$

indicating local Q-superlinear convergence. If in addition r is Lipschitz continuously differentiable in a neighborhood $\mathcal{N}$ *of* x_*; *that is,*

$$\|\nabla r(x_0) - \nabla r(x_1)\| \leqslant \beta_L \|x_0 - x_1\| \qquad \forall\, x_0, x_1 \in \mathcal{N}$$

for some $\beta_L > 0$, *we have for all* x_k *sufficiently close to* x_* *that*

$$\|x_{k+1} - x_*\| = \mathcal{O}(\|x_k - x_*\|^2), \tag{7}$$

indicating local Q-quadratic convergence.

§11.1 Local Algorithms

Proof.

Since $r(x_*) = 0$, we have from the previous theorem that

$$r(x_k) = r(x_k) - r(x_*) = J(x_k)(x_k - x_*) + w(x_k, x_*)\,,$$

where J is the Jacobian of r (that is, $J = \nabla r$) and

$$w(x_k, x_*) = \int_0^1 \big[J(x_* + t(x_k - x_*)) - J(x_k)\big](x_k - x_*)\, dt\,.$$

Note that Newton's direction p_k satisfies $J(x_k)p_k = -r_k$. By the fact that $J(x_k)$ is non-singular for sufficient large k, we have

$$-p_k = (x_k - x_*) + J(x_k)^{-1} w(x_k, x_*) \qquad \forall\, k \gg 1\,;$$

thus rearranging terms we obtain

$$x_k + p_k - x_* = -J(x_k)^{-1} w(x_k, x_*) \qquad \forall\, k \gg 1\,. \tag{8}$$

Next we estimate J^{-1} and w in order to conclude (6) and (7). □

§11.1 Local Algorithms

Proof.

Since $r(x_*) = 0$, we have from the previous theorem that

$$r(x_k) = r(x_k) - r(x_*) = J(x_k)(x_k - x_*) + w(x_k, x_*)\,,$$

where J is the Jacobian of r (that is, $J = \nabla r$) and

$$w(x_k, x_*) = \int_0^1 \big[J(x_* + t(x_k - x_*)) - J(x_k)\big](x_k - x_*)\, dt\,.$$

Note that Newton's direction p_k satisfies $J(x_k)p_k = -r_k$. By the fact that $J(x_k)$ is non-singular for sufficient large k, we have

$$-p_k = (x_k - x_*) + J(x_k)^{-1} w(x_k, x_*) \qquad \forall\, k \gg 1\,;$$

thus rearranging terms we obtain

$$x_k + p_k - x_* = -J(x_k)^{-1} w(x_k, x_*) \qquad \forall\, k \gg 1\,. \tag{8}$$

Next we estimate J^{-1} and w in order to conclude (6) and (7). □

§11.1 Local Algorithms

Proof.

Since $r(x_*) = 0$, we have from the previous theorem that

$$r(x_k) = r(x_k) - r(x_*) = J(x_k)(x_k - x_*) + w(x_k, x_*)\,,$$

where J is the Jacobian of r (that is, $J = \nabla r$) and

$$w(x_k, x_*) = \int_0^1 \big[J(x_* + t(x_k - x_*)) - J(x_k)\big](x_k - x_*)\, dt\,.$$

Note that Newton's direction p_k satisfies $J(x_k)p_k = -r_k$. By the fact that $J(x_k)$ is non-singular for sufficient large k, we have

$$-p_k = (x_k - x_*) + J(x_k)^{-1} w(x_k, x_*) \qquad \forall\, k \gg 1\,;$$

thus rearranging terms we obtain

$$x_k + p_k - x_* = -J(x_k)^{-1} w(x_k, x_*) \qquad \forall\, k \gg 1\,. \tag{8}$$

Next we estimate J^{-1} and w in order to conclude (6) and (7). □

§11.1 Local Algorithms

Proof (cont'd).

By the property of integrals,

$$\|w(x_k, x_*)\| = \left\| \int_0^1 \left[J(x_* + t(x_k - x_*)) - J(x_k) \right] (x_k - x_*) dt \right\|$$
$$\leqslant \int_0^1 \|J(x_* + t(x_k - x_*)) - J(x_k)\| \|x_k - x_*\| \, dt \, .$$

Therefore, by the continuous differentiability of r in $\mathcal{D}$,

$$\|w(x_k, x_*)\| = o(\|x_k - x_*\|) \, . \tag{9}$$

If in addition that r is Lipschitz continuously differentiable in a neighborhood of x_*, the inequality above implies that

$$\|w(x_k, x_*)\| \leqslant \int_0^1 \beta_L (1 - t) \|x_k - x_*\|^2 \, dt = \frac{\beta_L}{2} \|x_k - x_*\|^2 \quad \forall \, k \gg 1 \, ,$$

giving the result

$$\|w(x_k, x_*)\| = \mathcal{O}(\|x_k - x_*\|^2) \, . \tag{10}$$

§11.1 Local Algorithms

Proof (cont'd).

By the property of integrals,

$$\|w(x_k, x_*)\| = \left\| \int_0^1 \left[J(x_* + t(x_k - x_*)) - J(x_k) \right] (x_k - x_*) dt \right\|$$
$$\leqslant \int_0^1 \|J(x_* + t(x_k - x_*)) - J(x_k)\| \|x_k - x_*\| \, dt \, .$$

Therefore, by the continuous differentiability of r in $\mathcal{D}$,

$$\|w(x_k, x_*)\| = o(\|x_k - x_*\|) \, . \tag{9}$$

If in addition that r is Lipschitz continuously differentiable in a neighborhood of x_*, the inequality above implies that

$$\|w(x_k, x_*)\| \leqslant \int_0^1 \beta_L (1 - t) \|x_k - x_*\|^2 \, dt = \frac{\beta_L}{2} \|x_k - x_*\|^2 \quad \forall \, k \gg 1 \, ,$$

giving the result

$$\|w(x_k, x_*)\| = \mathcal{O}(\|x_k - x_*\|^2) \, . \tag{10}$$

§11.1 Local Algorithms

Proof (cont'd).

Moreover, since $J(x_*)$ is non-singular and $\mathcal{D}$ is open, there is a radius $\delta > 0$ and a positive constant β_* such that $B[x_*, \delta] \subseteq \mathcal{D}$ and

$$\|J(x)^{-1}\| \leqslant \beta_* \qquad \forall\, x \in B[x_*, \delta]\,. \tag{11}$$

Therefore, for $k \gg 1$ so that $x_k \in B[x_*, \delta]$ and $J(x_k)$ is non-singular, using (8), (9) and (11) we obtain

$$\|x_{k+1} - x_*\| = o(\|x_k - x_*\|)\,,$$

yielding desired estimate (6). Moreover, if in addition r is Lipschitz continuously differentiable in a neighborhood of x_*, using (8), (10) and (11) we obtain

$$\|x_{k+1} - x_*\| = \mathcal{O}(\|x_k - x_*\|^2)\,,$$

yielding desired estimate (7). □

§11.1 Local Algorithms

Proof (cont'd).

Moreover, since $J(x_*)$ is non-singular and $\mathcal{D}$ is open, there is a radius $\delta > 0$ and a positive constant β_* such that $B[x_*, \delta] \subseteq \mathcal{D}$ and

$$\|J(x)^{-1}\| \leqslant \beta_* \qquad \forall\, x \in B[x_*, \delta]\,. \tag{11}$$

Therefore, for $k \gg 1$ so that $x_k \in B[x_*, \delta]$ and $J(x_k)$ is non-singular, using (8), (9) and (11) we obtain

$$\|x_{k+1} - x_*\| = o(\|x_k - x_*\|)\,,$$

yielding desired estimate (6). Moreover, if in addition r is Lipschitz continuously differentiable in a neighborhood of x_*, using (8), (10) and (11) we obtain

$$\|x_{k+1} - x_*\| = \mathcal{O}(\|x_k - x_*\|^2)\,,$$

yielding desired estimate (7). □

§11.1 Local Algorithms

Proof (cont'd).

Moreover, since $J(x_*)$ is non-singular and $\mathcal{D}$ is open, there is a radius $\delta > 0$ and a positive constant β_* such that $B[x_*, \delta] \subseteq \mathcal{D}$ and

$$\|J(x)^{-1}\| \leqslant \beta_* \qquad \forall\, x \in B[x_*, \delta]\,. \tag{11}$$

Therefore, for $k \gg 1$ so that $x_k \in B[x_*, \delta]$ and $J(x_k)$ is non-singular, using (8), (9) and (11) we obtain

$$\|x_{k+1} - x_*\| = o(\|x_k - x_*\|)\,,$$

yielding desired estimate (6). Moreover, if in addition r is Lipschitz continuously differentiable in a neighborhood of x_*, using (8), (10) and (11) we obtain

$$\|x_{k+1} - x_*\| = \mathcal{O}(\|x_k - x_*\|^2)\,,$$

yielding desired estimate (7). □

§11.1 Local Algorithms

- **Inexact Newton method**

Instead of solving

$$J(x_k)p_k = -r(x_k) \tag{5}$$

exactly, inexact Newton methods use search directions p_k that satisfy the condition

$$\|r_k + J_k p_k\| \leqslant \eta_k \|r_k\| \quad \text{for some } \eta_k \in [0, \eta], \tag{12}$$

where $\eta \in [0, 1)$ is a constant. As in Chapter 7, we refer to $\{\eta_k\}$ as the forcing sequence. Different methods make different choices of the forcing sequence, and they use different algorithms for finding the approximate solutions p_k. The general framework for this class of methods can be stated as follows.

§11.1 Local Algorithms

Framework 11.2 (Inexact Newton for Nonlinear Equations).

Given $\eta \in [0, 1)$;

Choose x_0;

for $k = 0, 1, 2, \cdots$

Choose forcing parameter $\eta_k \in [0, \eta]$;

Find a vector p_k that satisfies

$$\|r_k + J_k p_k\| \leqslant \eta_k \|r_k\| \tag{12}$$

$x_{k+1} \leftarrow x_k + p_k$;

end (for)

§11.1 Local Algorithms

The convergence theory for these methods depends only on the condition (12) and not on the particular technique used to calculate p_k. The most important methods in this class, however, make use of iterative techniques for solving linear systems of the form $Jp = -r$, such as GMRES (Saad and Schultz [273], Walker [302]) or other Krylov space methods. Like the conjugate-gradient algorithm of Chapter 5 (which is not directly applicable here, since the coefficient matrix J is not symmetric positive definite), these methods typically require us to perform a matrix-vector multiplication of the form Jd for some d at each iteration, and to store a number of work vectors of length n. GMRES requires an additional vector to be stored at each iteration, so must be restarted periodically (often every 10 or 20 iterations) to keep memory requirements at a reasonable level.

§11.1 Local Algorithms

The matrix-vector products Jd can be computed without explicit knowledge of the Jacobian J. A finite-difference approximation to Jd that requires one evaluation of r is given by the formula (8.11). Calculation of Jd exactly (at least, to within the limits of finite-precision arithmetic) can be performed by using the forward mode of automatic differentiation, at a cost of at most a small multiple of an evaluation of r. Details of this procedure are given in Section 8.2. We do not discuss the iterative methods for sparse linear systems here, but refer the interested reader to Kelley [177] and Saad [272] for comprehensive descriptions and implementations of the most interesting techniques. We prove a local convergence theorem for the method, similar to the previous theorem.

§11.1 Local Algorithms

Theorem

Suppose that r is continuously differentiable in a convex open set $\mathcal{D} \subseteq \mathbb{R}^n$. Let $x_* \in \mathcal{D}$ be a non-degenerate solution of $r(x) = 0$, and let $\{x_k\}$ be the sequence of iterates generated by Framework 11.2. Then when $x_k \in \mathcal{D}$ is sufficiently close to x_*, the following statements are true:

1. *If η in Framework 11.2 is sufficiently small, the convergence of $\{x_k\}$ to x_* is Q-linear.*
2. *If $\eta_k \to 0$, the convergence is Q-superlinear.*
3. *If, in addition, J is Lipschitz continuous in a neighborhood of x_* and $\eta_k = \mathcal{O}(\|r_k\|)$, the convergence is Q-quadratic.*

§11.1 Local Algorithms

Proof.

We first rewrite (12) as

$$J(x_k)p_k + r(x_k) = v_k\,, \quad \text{where } \|v_k\| \leqslant \eta_k \|r(x_k)\|\,. \tag{13}$$

Since x_* is a non-degenerate root, we have as in (11) that there is a radius $\delta > 0$ and a constant β_* such that

$$\|J(x)^{-1}\| \leqslant \beta_* \qquad \forall\, x \in B[x_*, \delta]\,.$$

By multiplying both sides of (13) by $J(x_k)^{-1}$ and rearranging,

$$\|p_k + J(x_k)^{-1} r(x_k)\| = \|J(x_k)^{-1} v_k\| \leqslant \beta_* \eta_k \|r(x_k)\|\,. \tag{14}$$

As in the proof of the previous theorem, we have that

$$r(x) = J(x)(x - x_*) + w(x, x_*)\,, \tag{15}$$

where $\rho(x) \equiv \|w(x, x_*)\| / \|x - x_*\| \to 0$ as $x \to x_*$. □

§11.1 Local Algorithms

Proof.

We first rewrite (12) as

$$J(x_k)p_k + r(x_k) = v_k\,, \quad \text{where } \|v_k\| \leqslant \eta_k \|r(x_k)\|\,. \tag{13}$$

Since x_* is a non-degenerate root, we have as in (11) that there is a radius $\delta > 0$ and a constant β_* such that

$$\|J(x)^{-1}\| \leqslant \beta_* \qquad \forall\, x \in B[x_*, \delta]\,.$$

By multiplying both sides of (13) by $J(x_k)^{-1}$ and rearranging,

$$\|p_k + J(x_k)^{-1} r(x_k)\| = \|J(x_k)^{-1} v_k\| \leqslant \beta_* \eta_k \|r(x_k)\|\,. \tag{14}$$

As in the proof of the previous theorem, we have that

$$r(x) = J(x)(x - x_*) + w(x, x_*)\,, \tag{15}$$

where $\rho(x) \equiv \|w(x, x_*)\|/\|x - x_*\| \to 0$ as $x \to x_*$. □

§11.1 Local Algorithms

Proof.

We first rewrite (12) as

$$J(x_k)p_k + r(x_k) = v_k\,, \quad \text{where } \|v_k\| \leqslant \eta_k \|r(x_k)\|\,. \tag{13}$$

Since x_* is a non-degenerate root, we have as in (11) that there is a radius $\delta > 0$ and a constant β_* such that

$$\|J(x)^{-1}\| \leqslant \beta_* \qquad \forall\, x \in B[x_*, \delta]\,.$$

By multiplying both sides of (13) by $J(x_k)^{-1}$ and rearranging,

$$\|p_k + J(x_k)^{-1} r(x_k)\| = \|J(x_k)^{-1} v_k\| \leqslant \beta_* \eta_k \|r(x_k)\|\,. \tag{14}$$

As in the proof of the previous theorem, we have that

$$r(x) = J(x)(x - x_*) + w(x, x_*)\,, \tag{15}$$

where $\rho(x) \equiv \|w(x, x_*)\|/\|x - x_*\| \to 0$ as $x \to x_*$. □

§11.1 Local Algorithms

Proof (cont'd).

By reducing δ if necessary, we have from this expression that the following bound holds for all $x \in B[x_*, \delta]$:

$$\|r(x)\| \leqslant 2\|J(x_*)\|\|x - x_*\| + o(\|x - x_*\|) \leqslant 4\|J(x_*)\|\|x - x_*\|. \quad (16)$$

We now set $x = x_k$ in (15), and use (14) and (16) to obtain

$$\begin{aligned} \|x_k + p_k - x_*\| &= \|p_k + J(x_k)^{-1}(r(x_k) - w(x_k, x_*))\| \\ &\leqslant \beta_* \eta_k \|r(x_k)\| + \|J(x_k)^{-1}\|\|w(x_k, x_*)\| \\ &\leqslant \left[4\|J(x_*)\|\beta_* \eta_k + \beta_* \rho(x_k)\right]\|x_k - x_*\|. \quad (17) \end{aligned}$$

By choosing x_k close enough to x_* that $\rho(x_k) \leqslant 1/(4\beta_*)$, and choosing $\eta = 1/(16\|J(x_*)\|\beta_*)$, we have that the term in brackets in (17) is at most $1/2$. Hence, since $x_{k+1} = x_k + p_k$, this formula indicates Q-linear convergence of $\{x_k\}$ to x_*, proving ①. □

§11.1 Local Algorithms

Proof (cont'd).

By reducing δ if necessary, we have from this expression that the following bound holds for all $x \in B[x_*, \delta]$:

$$\|r(x)\| \leqslant 2\|J(x_*)\|\|x - x_*\| + o(\|x - x_*\|) \leqslant 4\|J(x_*)\|\|x - x_*\|. \quad (16)$$

We now set $x = x_k$ in (15), and use (14) and (16) to obtain

$$\begin{aligned} \|x_k + p_k - x_*\| &= \left\|p_k + J(x_k)^{-1}(r(x_k) - w(x_k, x_*))\right\| \\ &\leqslant \beta_*\eta_k\|r(x_k)\| + \left\|J(x_k)^{-1}\right\|\|w(x_k, x_*)\| \\ &\leqslant \left[4\|J(x_*)\|\beta_*\eta_k + \beta_*\rho(x_k)\right]\|x_k - x_*\|. \quad (17) \end{aligned}$$

By choosing x_k close enough to x_* that $\rho(x_k) \leqslant 1/(4\beta_*)$, and choosing $\eta = 1/(16\|J(x_*)\|\beta_*)$, we have that the term in brackets in (17) is at most $1/2$. Hence, since $x_{k+1} = x_k + p_k$, this formula indicates Q-linear convergence of $\{x_k\}$ to x_*, proving ①. □

§11.1 Local Algorithms

Proof (cont'd).

By reducing δ if necessary, we have from this expression that the following bound holds for all $x \in B[x_*, \delta]$:

$$\|r(x)\| \leqslant 2\|J(x_*)\|\|x - x_*\| + o(\|x - x_*\|) \leqslant 4\|J(x_*)\|\|x - x_*\|. \quad (16)$$

We now set $x = x_k$ in (15), and use (14) and (16) to obtain

$$\begin{aligned} \|x_k + p_k - x_*\| &= \left\|p_k + J(x_k)^{-1}(r(x_k) - w(x_k, x_*))\right\| \\ &\leqslant \beta_*\eta_k\|r(x_k)\| + \left\|J(x_k)^{-1}\right\|\|w(x_k, x_*)\| \\ &\leqslant \left[4\|J(x_*)\|\beta_*\eta_k + \beta_*\rho(x_k)\right]\|x_k - x_*\|. \quad (17) \end{aligned}$$

By choosing x_k close enough to x_* that $\rho(x_k) \leqslant 1/(4\beta_*)$, and choosing $\eta = 1/(16\|J(x_*)\|\beta_*)$, we have that the term in brackets in (17) is at most $1/2$. Hence, since $x_{k+1} = x_k + p_k$, this formula indicates Q-linear convergence of $\{x_k\}$ to x_*, proving ①. □

§11.1 Local Algorithms

Proof (cont'd).

Since $\rho(x_k) \to 0$ as $x_k \to x_*$, ② follows immediately from the fact that the term in brackets in

$$\|x_k + p_k - x_*\| \leqslant \left[4\|J(x_*)\|\beta_*\eta_k + \beta_*\rho(x_k)\right]\|x_k - x_*\| \tag{17}$$

approaches to zero as $x_k \to x_*$ and $\eta_k \to 0$. For ③, as in the proof of the previous theorem the Lipschitz continuous differentiability of r implies that

$$\|w(x_k, x_*)\| = \mathcal{O}(\|x_k - x_*\|^2)\,.$$

If $\eta_k = \mathcal{O}(\|r_k\|)$, using (16) we conclude that $\eta_k = \mathcal{O}(\|x_k - x_*\|)$; thus (17) shows that

$$\|x_{k+1} - x_*\| = \mathcal{O}(\|x_k - x_*\|^2)\,;$$

thus proving quadratic convergence. □

§11.1 Local Algorithms

Proof (cont'd).

Since $\rho(x_k) \to 0$ as $x_k \to x_*$, ② follows immediately from the fact that the term in brackets in

$$\|x_k + p_k - x_*\| \leqslant \left[4\|J(x_*)\|\beta_*\eta_k + \beta_*\rho(x_k)\right]\|x_k - x_*\| \qquad (17)$$

approaches to zero as $x_k \to x_*$ and $\eta_k \to 0$. For ③, as in the proof of the previous theorem the Lipschitz continuous differentiability of r implies that

$$\|w(x_k, x_*)\| = \mathcal{O}(\|x_k - x_*\|^2)\,.$$

If $\eta_k = \mathcal{O}(\|r_k\|)$, using (16) we conclude that $\eta_k = \mathcal{O}(\|x_k - x_*\|)$; thus (17) shows that

$$\|x_{k+1} - x_*\| = \mathcal{O}(\|x_k - x_*\|^2)\,;$$

thus proving quadratic convergence. □

§11.1 Local Algorithms

- **Broyden's method**

Secant methods, also known as quasi-Newton methods, do not require calculation of the Jacobian $J(x)$. Instead, they construct their own approximation to this matrix, updating it at each iteration so that it mimics the behavior of the true Jacobian J over the step just taken. The approximate Jacobian, which we denote at iteration k by B_k, is then used to construct a linear model analogous to (4), namely

$$M_k(p) = r(x_k) + B_k p\,. \tag{18}$$

We obtain the step by setting this model to zero. When B_k is nonsingular, we have the following explicit formula (cf. (5)):

$$p_k = -B_k^{-1} r(x_k)\,. \tag{19}$$

§11.1 Local Algorithms

The requirement that the approximate Jacobian should mimic the behavior of the true Jacobian can be specified as follows. Let s_k and y_k be defined by

$$s_k = x_{k+1} - x_k\,, \quad y_k = r(x_{k+1}) - r(x_k)\,. \tag{20}$$

From Taylor's Theorem, s_k and y_k are related by the expression

$$y_k = \int_0^1 J(x_k + ts_k)s_k dt \approx J(x_{k+1})s_k + o(\|s_k\|)\,. \tag{21}$$

We require the updated Jacobian approximation B_{k+1} to satisfy the following equation, which is known as the secant equation,

$$y_k = B_{k+1}s_k\,, \tag{22}$$

which ensures that B_{k+1} and $J(x_{k+1})$ have similar behavior along the direction s_k.

§11.1 Local Algorithms

The secant equation does not say anything about how B_{k+1} should behave along directions orthogonal to s_k. In fact, we can view (22) as a system of n linear equations in n^2 unknowns, where the unknowns are the components of B_{k+1}, so for $n > 1$ the equation (22) does not determine all the components of B_{k+1} uniquely (the scalar case of $n = 1$ gives rise to the scalar secant method). The most successful practical algorithm is Broyden's method, for which the update formula is

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k}. \tag{23}$$

The Broyden update makes the smallest possible change to the Jacobian (as measured by the Euclidean norm $\|B_k - B_{k+1}\|^2$) that is consistent with (22), as we show in the lemma in the next slide.

§11.1 Local Algorithms

Lemma

Among all matrices B satisfying $Bs_k = y_k$, matrix B_{k+1} defined by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k} \tag{23}$$

minimizes the difference $\|B - B_k\|$.

Proof.

Let B be any matrix that satisfies $Bs_k = y_k$. By the fact that $\|ss^{\mathrm{T}}/s^{\mathrm{T}}s\| = 1$ for any vector s, we have

$$\|B_{k+1} - B_k\| = \left\| \frac{(y_k - B_k s_k) s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k} \right\| = \left\| \frac{(B - B_k) s_k s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k} \right\|$$

$$\leqslant \|B - B_k\| \left\| \frac{s_k s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k} \right\| = \|B - B_k\|.$$

Therefore, $B_{k+1} \in \underset{B\,:\,y_k = Bs_k}{\operatorname{argmin}} \|B - B_k\|$, and the result is proved. □

§11.1 Local Algorithms

Lemma

Among all matrices B satisfying $Bs_k = y_k$, matrix B_{k+1} defined by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k} \tag{23}$$

minimizes the difference $\|B - B_k\|$.

Proof.

Let B be any matrix that satisfies $Bs_k = y_k$. By the fact that $\left\|ss^{\mathrm{T}}/s^{\mathrm{T}}s\right\| = 1$ for any vector s, we have

$$\begin{aligned}\|B_{k+1} - B_k\| &= \left\|\frac{(y_k - B_k s_k)s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k}\right\| = \left\|\frac{(B - B_k)s_k s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k}\right\| \\ &\leqslant \|B - B_k\|\left\|\frac{s_k s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k}\right\| = \|B - B_k\|.\end{aligned}$$

Therefore, $B_{k+1} \in \underset{B\,:\,y_k = Bs_k}{\operatorname{argmin}} \|B - B_k\|$, and the result is proved. □

§11.1 Local Algorithms

In the specification of the algorithm below, we allow a line search to be performed along the search direction p_k, so that $s_k = \alpha p_k$ for some $\alpha > 0$ in the formula (20).

Algorithm 11.3 (Broyden).

Choose x_0 and a non-singular initial Jacobian approximation B_0;

for $k = 0, 1, 2, \cdots$

Calculate a solution p_k to the linear equations

$$B_k p_k = -r(x_k); \tag{24}$$

Choose α_k by performing a line search along p_k;

$x_{k+1} \leftarrow x_k + \alpha_k p_k$;

$s_k \leftarrow x_{k+1} - x_k$;

$y_k \leftarrow r(x_{k+1}) - r(x_k)$;

Obtain B_{k+1} from the formula (23);

end (for)

§11.1 Local Algorithms

In the specification of the algorithm below, we allow a line search to be performed along the search direction p_k, so that $s_k = \alpha p_k$ for some $\alpha > 0$ in the formula (20).

Algorithm 11.3 (Broyden).

Choose x_0 and a non-singular initial Jacobian approximation B_0;

for $k = 0, 1, 2, \cdots$

Calculate a solution p_k to the linear equations

$$B_k p_k = -r(x_k); \tag{24}$$

Choose α_k by performing a line search along p_k;

$x_{k+1} \leftarrow x_k + \alpha_k p_k$;

$s_k \leftarrow x_{k+1} - x_k$;

$y_k \leftarrow r(x_{k+1}) - r(x_k)$;

Obtain B_{k+1} from the formula (23);

end (for)

§11.1 Local Algorithms

Under certain assumptions, Broyden's method converges superlinearly. This local convergence rate is fast enough for most practical purposes, though not as fast as the Q-quadratic convergence of Newton's method. We illustrate the difference between the convergence rates of Newton's and Broyden's method with a small example. The function $r : \mathbb{R}^2 \to \mathbb{R}^2$ defined by

$$r(x) = \begin{bmatrix} (x_1 + 3)(x_2^3 - 7) + 18 \\ \sin(x_2 e^{x_1} - 1) \end{bmatrix} \tag{25}$$

has a non-degenerate root at $x_* = (0, 1)^{\mathrm{T}}$. We start both methods from the point $x_0 = (-0.5, 1.4)^{\mathrm{T}}$, and use the exact Jacobian $J(x_0)$ at this point as the initial Jacobian approximation B_0. Results are shown in Table 11.1.

§11.1 Local Algorithms

Newton's method clearly exhibits Q-quadratic convergence, which is characterized by doubling of the exponent of the error at each iteration. Broyden's method takes twice as many iterations as Newton's, and reduces the error at a rate that accelerates slightly towards the end.

Table 11.1 Convergence of Iterates in Broyden and Newton Methods

	$\|x_k - x^*\|_2$	
Iteration k	Broyden	Newton
0	0.64×10^{0}	0.64×10^{0}
1	0.62×10^{-1}	0.62×10^{-1}
2	0.52×10^{-3}	0.21×10^{-3}
3	0.25×10^{-3}	0.18×10^{-7}
4	0.43×10^{-4}	0.12×10^{-15}
5	0.14×10^{-6}	
6	0.57×10^{-9}	
7	0.18×10^{-11}	
8	0.87×10^{-15}	

Table 11.2 Convergence of Function Norms in Broyden and Newton Methods

	$\|r(x_k)\|_2$	
Iteration k	Broyden	Newton
0	0.74×10^{1}	0.74×10^{1}
1	0.59×10^{0}	0.59×10^{0}
2	0.20×10^{-2}	0.23×10^{-2}
3	0.21×10^{-2}	0.16×10^{-6}
4	0.37×10^{-3}	0.22×10^{-15}
5	0.12×10^{-5}	
6	0.49×10^{-8}	
7	0.15×10^{-10}	
8	0.11×10^{-18}	

§11.1 Local Algorithms

The function norms $\|r(x_k)\|$ approach zero at a similar rate to the iteration errors $\|x_k - x_*\|$. As in

$$\begin{aligned} r(x_k) &= r(x_k) - r(x_*) \\ &= J(x_k)(x_k - x_*) + \int_0^1 \big[J(x_* + t(x_k - x_*)) - J(x_k)\big](x_k - x_*)\, dt\,, \end{aligned}$$

we have that

$$r(x_k) = r(x_k) - r(x_*) \approx J(x_*)(x_k - x_*)\,,$$

so by non-singularity of $J(x_*)$, the norms of $r(x_k)$ and $(x_k - x_*)$ are bounded above and below by multiples of each other. For our example problem (25), convergence of the sequence of function norms in the two methods is shown in Table 11.2.

§11.1 Local Algorithms

The convergence analysis of Broyden's method is more complicated than that of Newton's method. We state the following result without proof.

Theorem

Suppose that r is continuously differentiable in a convex open set $\mathcal{D} \subseteq \mathbb{R}^n$. Let $x_* \in \mathcal{D}$ be a non-degenerate solution of $r(x) = 0$. Then there are positive constants ε and δ such that if the starting point x_0 and the starting approximate Jacobian B_0 satisfy

$$\|x_0 - x_*\| \leqslant \delta \quad \text{and} \quad \|B_0 - J(x_*)\| \leqslant \varepsilon\,, \tag{26}$$

the sequence $\{x_k\}$ generated by Broyden's method is well-defined and converges Q-superlinearly to x_*.

§11.1 Local Algorithms

The second condition $\|B_0 - J(x_*)\| \leqslant \varepsilon$ in (26) is difficult to guarantee in practice. In contrast to the case of unconstrained minimization, a good choice of B_0 can be critical to the performance of the algorithm. Some implementations of Broyden's method recommend choosing B_0 to be $J(x_0)$, or some finite-difference approximation to this matrix.

The Broyden matrix B_k will be dense in general, even if the true Jacobian J is sparse. Therefore, when n is large, an implementation of Broyden's method that stores B_k as a full $n \times n$ matrix may be inefficient. Instead, we can use limited-memory methods in which B_k is stored implicitly in the form of a number of vectors of length n, while the system (24) is solved by a technique based on application of the Sherman-Morrison-Woodbury formula.

§11.1 Local Algorithms

The second condition $\|B_0 - J(x_*)\| \leqslant \varepsilon$ in (26) is difficult to guarantee in practice. In contrast to the case of unconstrained minimization, a good choice of B_0 can be critical to the performance of the algorithm. Some implementations of Broyden's method recommend choosing B_0 to be $J(x_0)$, or some finite-difference approximation to this matrix.

The Broyden matrix B_k will be dense in general, even if the true Jacobian J is sparse. Therefore, when n is large, an implementation of Broyden's method that stores B_k as a full $n \times n$ matrix may be inefficient. Instead, we can use limited-memory methods in which B_k is stored implicitly in the form of a number of vectors of length n, while the system (24) is solved by a technique based on application of the Sherman-Morrison-Woodbury formula.

§11.1 Local Algorithms

- **Tensor methods**

In tensor methods, the linear model $M_k(p)$ used by Newton's method (4) is augmented with an extra term that aims to capture some of the nonlinear, higher-order, behavior of r. By doing so, it achieves more rapid and reliable convergence to degenerate roots, in particular, to roots x_* for which the Jacobian $J(x_*)$ has rank $n-1$ or $n-2$. We give a broad outline of the method here, and refer to Schnabel and Frank [277] for details.

§11.1 Local Algorithms

We use $\widehat{M}_k(p)$ to denote the model function on which tensor methods are based; this function has the form

$$\widehat{M}_k(p) = r(x_k) + J(x_k)p + \frac{1}{2}T_k(p,p)\,, \tag{27}$$

where T_k is a tensor defined by n^3 elements $(T_k)_{ij\ell}$ whose action on a pair of arbitrary vectors u and v in $\mathbb{R}^n$ is defined by

$$\begin{aligned}\big[T_k(u,v)\big]_i &= \text{the } i\text{-th component of } T_k(u,v)\\ &= \sum_{j=1}^{n}\sum_{\ell=1}^{n}(T_k)_{ij\ell}u_j v_\ell\,.\end{aligned}$$

If we followed the reasoning behind Newton's method, we could consider building T_k from the second derivatives of r at the point x_k; that is,

$$(T_k)_{ij\ell} = \big[\nabla^2 r_i(x_k)\big]_{j\ell}\,.$$

§11.1 Local Algorithms

However, use of the exact second derivatives is not practical in most instances. If we were to store this information explicitly, about $n^3/2$ memory locations would be needed, about n times the requirements of Newton's method. Moreover, there may be no vector p for which $\widehat{M}_k(p) = 0$, so the step may not even be defined.

Instead, the approach described in [277] defines T_k in a way that requires little additional storage, but which gives $\widehat{M}_k$ some potentially appealing properties. Specifically, T_k is chosen so that $\widehat{M}_k(p)$ interpolates the function $r(x_k + p)$ at some previous iterates visited by the algorithm. That is, we require that

$$\widehat{M}_k(x_{k-j} - x_k) = r(x_{k-j}) \quad \text{for } j = 1, 2, \cdots, q, \tag{28}$$

for some integer $q > 0$.

§11.1 Local Algorithms

By substituting from (27), we see that T_k must satisfy the condition

$$\frac{1}{2} T_k(s_{jk}, s_{jk}) = r(x_{k-j}) - r(x_k) - J(x_k)s_{jk},$$

where

$$s_{jk} \equiv x_{k-j} - x_k \,, \text{for } j = 1,\ 2,\ \cdots,\ q.$$

In [277] it is shown that this condition can be ensured by choosing T_k so that its action on arbitrary vectors u and v is

$$T_k(u, v) = \sum_{j=1}^{q} a_j (s_{jk}^{\mathrm{T}} u)(s_{jk}^{\mathrm{T}} v) \,,$$

where a_j, $j = 1,\ 2,\ \cdots,\ q$, are vectors of length n. The number of interpolating points q is typically chosen to be quite modest, usually less than $\sqrt{n}$. This T_k can be stored in $2nq$ locations, which contain the vectors a_j and s_{jk} for $j = 1,\ 2,\ \cdots,\ q$.

§11.1 Local Algorithms

This technique can be refined in various ways. The points of interpolation can be chosen to make the collection of directions s_{jk} more linearly independent. There may still not be a vector p for which $\widehat{M}_k(p) = 0$, but we can instead take the step to be the vector that minimizes $\|\widehat{M}_k(p)\|^2$, which can be found by using a specialized least-squares technique. There is no assurance that the step obtained in this way is a descent direction for the merit function $\frac{1}{2}\|r(x)\|^2$ (which is discussed in the next section), and in this case it can be replaced by the standard Newton direction $-J_k^{-1}r_k$.

§11.2 Practical Methods

We now consider practical variants of the Newton-like methods discussed above, in which line-search and trust-region modifications to the steps are made in order to ensure better global convergence behavior.

§11.2 Practical Methods

- **Merit functions**

As mentioned above, neither Newton's method (5) nor Broyden's method (19), (23) with unit step lengths can be guaranteed to converge to a solution of $r(x) = 0$ unless they are started close to that solution. Sometimes, components of the unknown or function vector or the Jacobian will blow up. Another, more exotic, kind of behavior is cycling, where the iterates move between distinct regions of the parameter space without approaching a root. An example is the scalar function

$$r(x) = -x^5 + x^3 + 4x,$$

which has five non-degenerate roots. When started fromthe point $x_0 = 1$, Newton's method produces a sequence of iterates that oscillates between 1 and -1 without converging to any of the roots.

§11.2 Practical Methods

The Newton and Broyden methods can be made more robust by using line-search and trust-region techniques similar to those described in Chapters 3 and 4. Before describing these techniques, we need to define a merit function, which is a scalar-valued function of x that indicates whether a new iterate is better or worse than the current iterate, in the sense of making progress toward a root of r. In unconstrained optimization, the objective function f is itself a natural merit function; most algorithms for minimizing f require a decrease in f at each iteration. In nonlinear equations, the merit function is obtained by combining the n components of the vector r in some way.

§11.2 Practical Methods

The most widely used merit function is the sum of squares, defined by

$$f(x) = \frac{1}{2}\|r(x)\|^2 = \frac{1}{2}\sum_{j=1}^{n} r_j^2(x)\,. \tag{29}$$

Here we remark that the factor $1/2$ is introduced for convenience. Any root x_* of r obviously has $f(x_*) = 0$, and since $f(x) \geqslant 0$ for all x, each root is a minimizer of f. However, local minimizers of f are not roots of r if f is strictly positive at the point in question. Still, the merit function given by (29) has been used successfully in many applications and is implemented in a number of software packages.

§11.2 Practical Methods

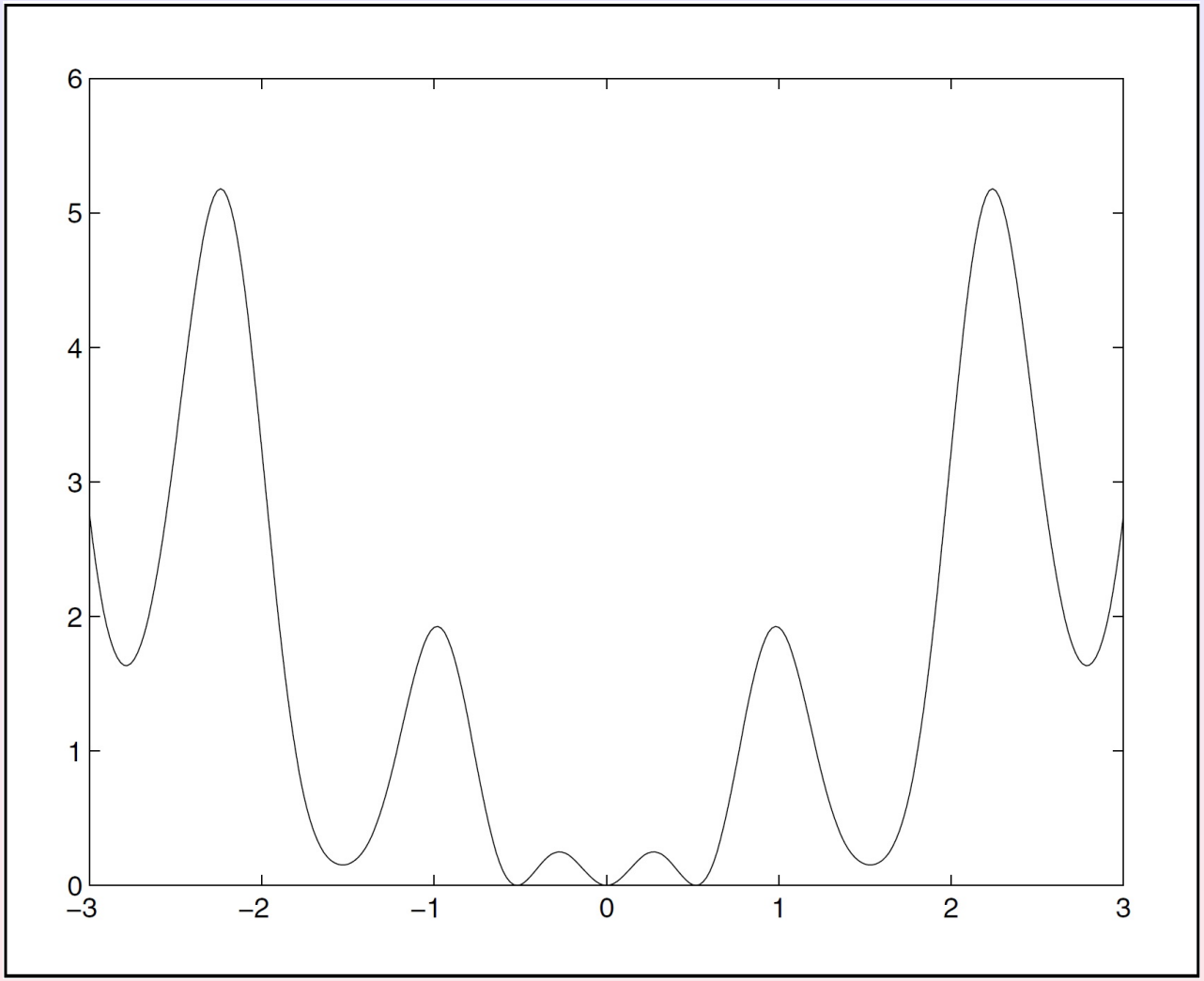

Figure 2: Plot of $\frac{1}{2}\big[\sin(5x)-x\big]^2$, showing its many local minima.

§11.2 Practical Methods

The merit function for the example $r(x) = \sin(5x) - x$ is plotted in Figure 2. It shows three local minima corresponding to the three roots, but there are many other local minima (for example, those at around ± 1.53053). Local minima like these that are not roots of f satisfy an interesting property. Since

$$\nabla f(x_*) = J(x_*)^{\mathrm{T}} r(x_*) = 0\,, \tag{30}$$

we can have $r(x_*) \neq 0$ only if $J(x_*)$ is singular.

Since local minima for the sum-of-squares merit function may be points of attraction for the algorithms described in this section, global convergence results for the algorithms discussed here are less satisfactory than for similar algorithms applied to unconstrained optimization.

§11.2 Practical Methods

Other merit functions are also used in practice. One such is the ℓ_1 norm merit function defined by

$$f_1(x) = \|r(x)\|_1 = \sum_{j=1}^{m} |r_j(x)| .$$

This function is studied in Chapters 17 and 18 in the context of algorithms for constrained optimization.

In the rest of Section 11.2, the minimization algorithms introduced in Chapter 3 and 4, including line search method and trust-region methods, are used to solve for nonlinear equations (or zeros of merit functions). The theory behind these algorithms are more or less identical to those we have seen in Chapter 3 and 4, and we omit here.

§11.2 Practical Methods

Other merit functions are also used in practice. One such is the ℓ_1 norm merit function defined by

$$f_1(x) = \|r(x)\|_1 = \sum_{j=1}^{m} |r_j(x)| .$$

This function is studied in Chapters 17 and 18 in the context of algorithms for constrained optimization.

In the rest of Section 11.2, the minimization algorithms introduced in Chapter 3 and 4, including line search method and trust-region methods, are used to solve for nonlinear equations (or zeros of merit functions). The theory behind these algorithms are more or less identical to those we have seen in Chapter 3 and 4, and we omit here.

§11.3 Continuation/Homotopy Methods

- **Motivation**

如前所述，基於牛頓的方法都有一個缺點：除非在感興趣的區域內 $J(x)$ 是 non-singular 的，否則它們有可能收斂到 merit function 的相對極小值，而不是非線性系統的解。在本節中我們介紹的連續法更有可能在複雜情況下收斂到 $r(x) = 0$ 的解。它們的基本動機很容易描述：與其直接處理原始問題 $r(x) = 0$，我們建立一個相對簡單的方程系統（求其解是簡單的）。然後，我們逐漸將這個簡單系統轉換為原始系統 $r(x)$，並隨著解從簡單問題的解移動到原始問題的解。

§11.3 Continuation/Homotopy Methods

One simple way to define the so-called homotopy map $H(x, \lambda)$ is as follows:

$$H(x, \lambda) = \lambda r(x) + (1 - \lambda)(x - a), \tag{31}$$

where λ is a scalar parameter and $a \in \mathbb{R}^n$ is a fixed vector. When $\lambda = 0$, (31) defines the artificial, easy problem $H(x, 0) = x - a$, whose solution is obviously $x = a$. When $\lambda = 1$, we have $H(x, 1) = r(x)$, the original system of equations. To solve $r(x) = 0$, consider the following algorithm: First, set $\lambda = 0$ in (31) and set $x = a$. Then, increase λ from 0 to 1 in small increments, and for each value of λ, calculate the solution of the system $H(x, \lambda) = 0$. The final value of x corresponding to $\lambda = 1$ will solve the original problem $r(x) = 0$.

§11.3 Continuation/Homotopy Methods

One simple way to define the so-called homotopy map $H(x, \lambda)$ is as follows:

$$H(x, \lambda) = \lambda r(x) + (1 - \lambda)(x - a), \tag{31}$$

where λ is a scalar parameter and $a \in \mathbb{R}^n$ is a fixed vector. When $\lambda = 0$, (31) defines the artificial, easy problem $H(x, 0) = x - a$, whose solution is obviously $x = a$. When $\lambda = 1$, we have $H(x, 1) = r(x)$, the original system of equations. To solve $r(x) = 0$, consider the following algorithm: First, set $\lambda = 0$ in (31) and set $x = a$. Then, increase λ from 0 to 1 in small increments, and for each value of λ, calculate the solution of the system $H(x, \lambda) = 0$. The final value of x corresponding to $\lambda = 1$ will solve the original problem $r(x) = 0$.

§11.3 Continuation/Homotopy Methods

This naive approach sounds plausible, and Figure 3 illustrates a situation in which it would be successful. In this figure, there is a unique solution x of the system $H(x, \lambda) = 0$ for each value of λ in the range $[0, 1]$. The trajectory of points (x, λ) for which $H(x, \lambda) = 0$ is called the **zero path**.

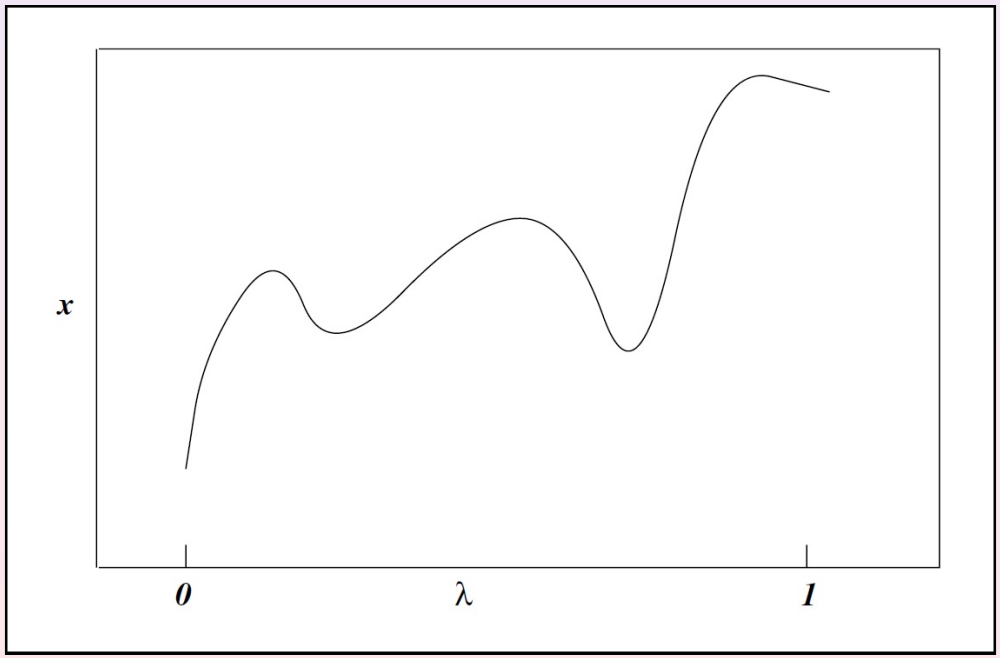

Figure 3: Plot of a zero path: Trajectory of points (x, λ) with $H(x, \lambda) = 0$.

§11.3 Continuation/Homotopy Methods

Unfortunately, however, the approach often fails, as illustrated in Figure 4. Here, the algorithm follows the lower branch of the curve from $\lambda = 0$ to $\lambda = \lambda_T$, but it then loses the trail unless it is lucky enough to jump to the top branch of the path. The value λ_T is known as a turning point, since at this point we can follow the path smoothly only if we no longer insist on increasing λ at every step. In fact, practical continuation methods work by doing exactly as Figure 4 (in the next slide) suggests; that is, they follow the zero path explicitly, even if this means allowing λ to decrease from time to time.

§11.3 Continuation/Homotopy Methods

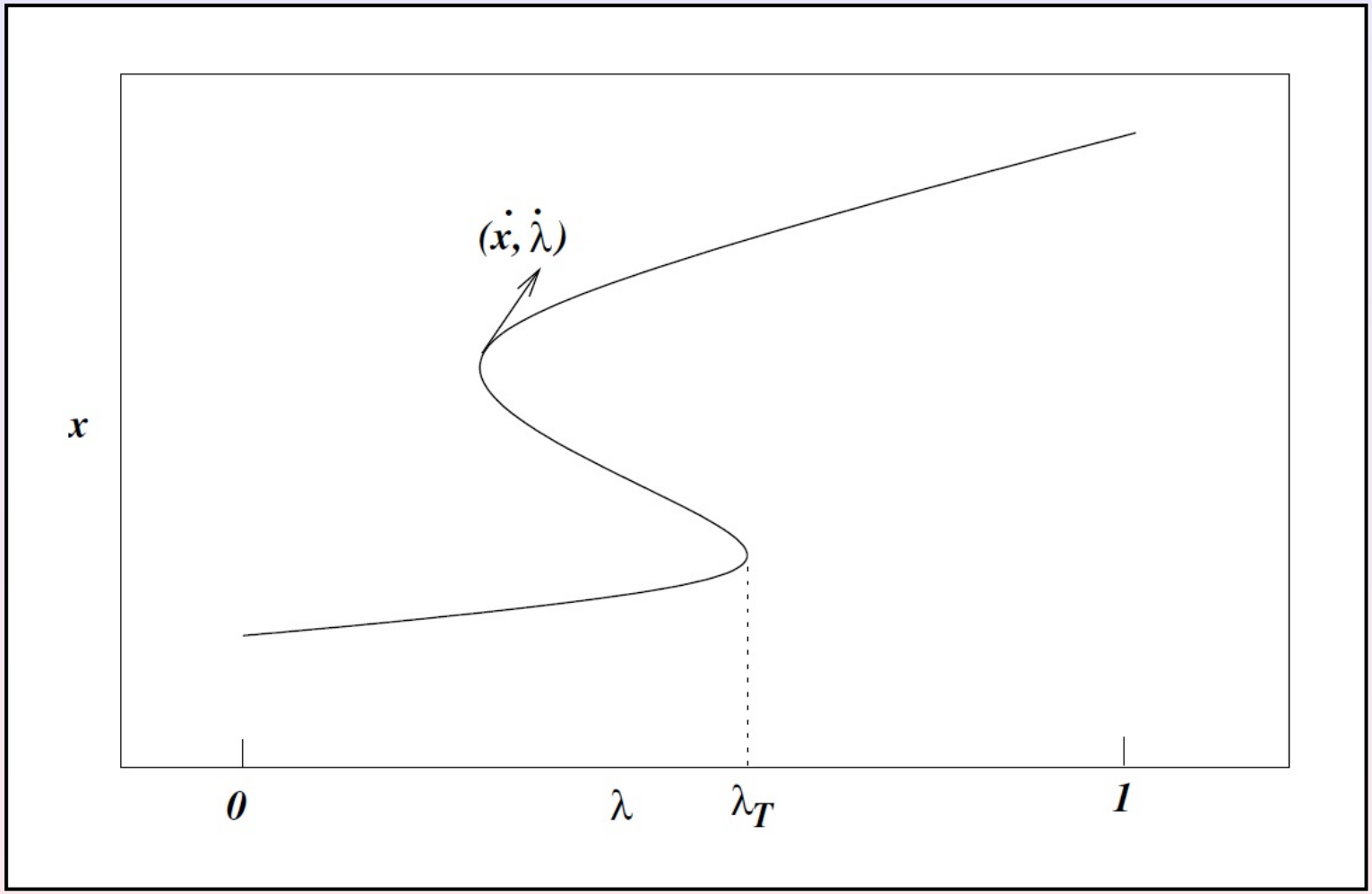

Figure 4: Zero path with turning points. The path joining $(a, 0)$ to $(x_*, 1)$ cannot be followed by increasing λ monotonically from 0 to 1.

§11.3 Continuation/Homotopy Methods

- **Practical continuation methods**

In one practical technique, we model the zero path by allowing both x and λ to be functions of an independent variable s that represents **arc length** along the path; that is, $(x(s), \lambda(s))$ is the point that we arrive at by traveling a distance s along the path from the initial point $(x(0), \lambda(0)) = (a, 0)$. Because we have that

$$H(x(s), \lambda(s)) = 0 \quad \forall\, s \geqslant 0\,,$$

we can take the total derivative of this expression with respect to s to obtain

$$\frac{\partial}{\partial x} H(x, \lambda)\dot{x} + \frac{\partial}{\partial \lambda} H(x, \lambda)\dot{\lambda} = 0\,, \quad \text{where } (\dot{x}, \dot{\lambda}) = \left(\frac{dx}{ds}, \frac{d\lambda}{ds}\right). \quad (32)$$

§11.3 Continuation/Homotopy Methods

- **Practical continuation methods**

In one practical technique, we model the zero path by allowing both x and λ to be functions of an independent variable s that represents **arc length** along the path; that is, $(x(s), \lambda(s))$ is the point that we arrive at by traveling a distance s along the path from the initial point $(x(0), \lambda(0)) = (a, 0)$. Because we have that

$$H(x(s), \lambda(s)) = 0 \quad \forall\, s \geqslant 0\,,$$

we can take the total derivative of this expression with respect to s to obtain

$$\frac{\partial}{\partial x} H(x, \lambda)\dot{x} + \frac{\partial}{\partial \lambda} H(x, \lambda)\dot{\lambda} = 0\,, \quad \text{where } (\dot{x}, \dot{\lambda}) = \left(\frac{dx}{ds}, \frac{d\lambda}{ds}\right). \quad (32)$$

§11.3 Continuation/Homotopy Methods

The vector $(\dot{x}(s), \dot{\lambda}(s))$ is the tangent vector to the zero path, as we illustrate in Figure 4. From (32), we see that this vector lies in the null space of the $n \times (n+1)$ matrix

$$\begin{bmatrix} \frac{\partial}{\partial x} H(x, \lambda) & \frac{\partial}{\partial \lambda} H(x, \lambda) \end{bmatrix}. \tag{33}$$

When this matrix has full rank, its null space has dimension 1, so to complete the definition of $(\dot{x}, \dot{\lambda})$ in this case, we need to assign it a length and direction. The length is fixed by imposing the normalization condition

$$\|\dot{x}(s)\|^2 + \|\dot{\lambda}(s)\|^2 = 1 \quad \forall\, s \geqslant 0 \tag{34}$$

which ensures that s is the true arc length along the path from $(0, a)$ to $(x(s), \lambda(s))$.

§11.3 Continuation/Homotopy Methods

The vector $(\dot{x}(s), \dot{\lambda}(s))$ is the tangent vector to the zero path, as we illustrate in Figure 4. From (32), we see that this vector lies in the null space of the $n \times (n+1)$ matrix

$$\left[\begin{array}{cc} \dfrac{\partial}{\partial x} H(x, \lambda) & \dfrac{\partial}{\partial \lambda} H(x, \lambda) \end{array} \right]. \tag{33}$$

When this matrix has full rank, its null space has dimension 1, so to complete the definition of $(\dot{x}, \dot{\lambda})$ in this case, we need to assign it a length and direction. The length is fixed by imposing the normalization condition

$$\|\dot{x}(s)\|^2 + \|\dot{\lambda}(s)\|^2 = 1 \quad \forall\, s \geqslant 0 \tag{34}$$

which ensures that s is the true arc length along the path from $(0, a)$ to $(x(s), \lambda(s))$.

§11.3 Continuation/Homotopy Methods

We need to choose the sign to ensure that we keep moving forward along the zero path. A heuristic that works well is to choose the sign so that the tangent vector $(\dot{x}, \dot{\lambda})$ at the current value of s makes an angle of less than $\pi/2$ with the tangent point at the previous value of s.

§11.3 Continuation/Homotopy Methods

We can outline the complete procedure for computing $(\dot{x}, \dot{\lambda})$ as follows:

Procedure 11.7 (Tangent Vector Calculation).

Compute a vector in the null space of (33) by performing a QR factorization with column pivoting,

$$Q^{\mathrm{T}} \begin{bmatrix} \frac{\partial}{\partial x} H(x,\lambda) & \frac{\partial}{\partial \lambda} H(x,\lambda) \end{bmatrix} \Pi = \begin{bmatrix} R & w \end{bmatrix},$$

where Q is $n \times n$ orthogonal, R is $n \times n$ upper triangular, Π is an $(n+1) \times (n+1)$ permutation matrix, and $w \in \mathbb{R}^n$.

Set

$$v = \Pi \begin{bmatrix} R^{-1} w \\ -1 \end{bmatrix};$$

Set $(\dot{x}, \dot{\lambda}) = \pm v / \|v\|$, where the sign is chosen to satisfy the angle criterion mentioned above.

§11.3 Continuation/Homotopy Methods

Since we can obtain the tangent at any given point (x, λ) and since we know the initial point $(x(0), \lambda(0)) = (a, 0)$, we can trace the zero path by calling a standard initial-value first-order ordinary differential equation solver (such as ode45 in matlab®), terminating the algorithm when it finds a value of s for which $\lambda(s) = 1$.

§11.3 Continuation/Homotopy Methods

A second approach for following the zero path is quite similar to the one just described, except that it takes an **algebraic viewpoint** instead of a differential-equations viewpoint. Given a current point (x, λ), we compute the tangent vector $(\dot{x}, \dot{\lambda})$ as above, and take a small step (of length ε, say) along this direction to produce a "predictor" point (x^P, λ^P); that is,

$$(x^P, \lambda^P) = (x, \lambda) + \varepsilon(\dot{x}, \dot{\lambda}) .$$

Usually, this new point will not lie exactly on the zero path, so we apply some "corrector" iterations to bring it back to the path, thereby identifying a new iterate (x^+, λ^+) that satisfies $H(x^+, \lambda^+) = 0$. This process is illustrated in Figure 5 in the next slide.

§11.3 Continuation/Homotopy Methods

A second approach for following the zero path is quite similar to the one just described, except that it takes an **algebraic viewpoint** instead of a differential-equations viewpoint. Given a current point (x, λ), we compute the tangent vector $(\dot{x}, \dot{\lambda})$ as above, and take a small step (of length ε, say) along this direction to produce a "predictor" point (x^P, λ^P); that is,

$$(x^P, \lambda^P) = (x, \lambda) + \varepsilon(\dot{x}, \dot{\lambda}) .$$

Usually, this new point will not lie exactly on the zero path, so we apply some "corrector" iterations to bring it back to the path, thereby identifying a new iterate (x^+, λ^+) that satisfies $H(x^+, \lambda^+) = 0$. This process is illustrated in Figure 5 in the next slide.

§11.3 Continuation/Homotopy Methods

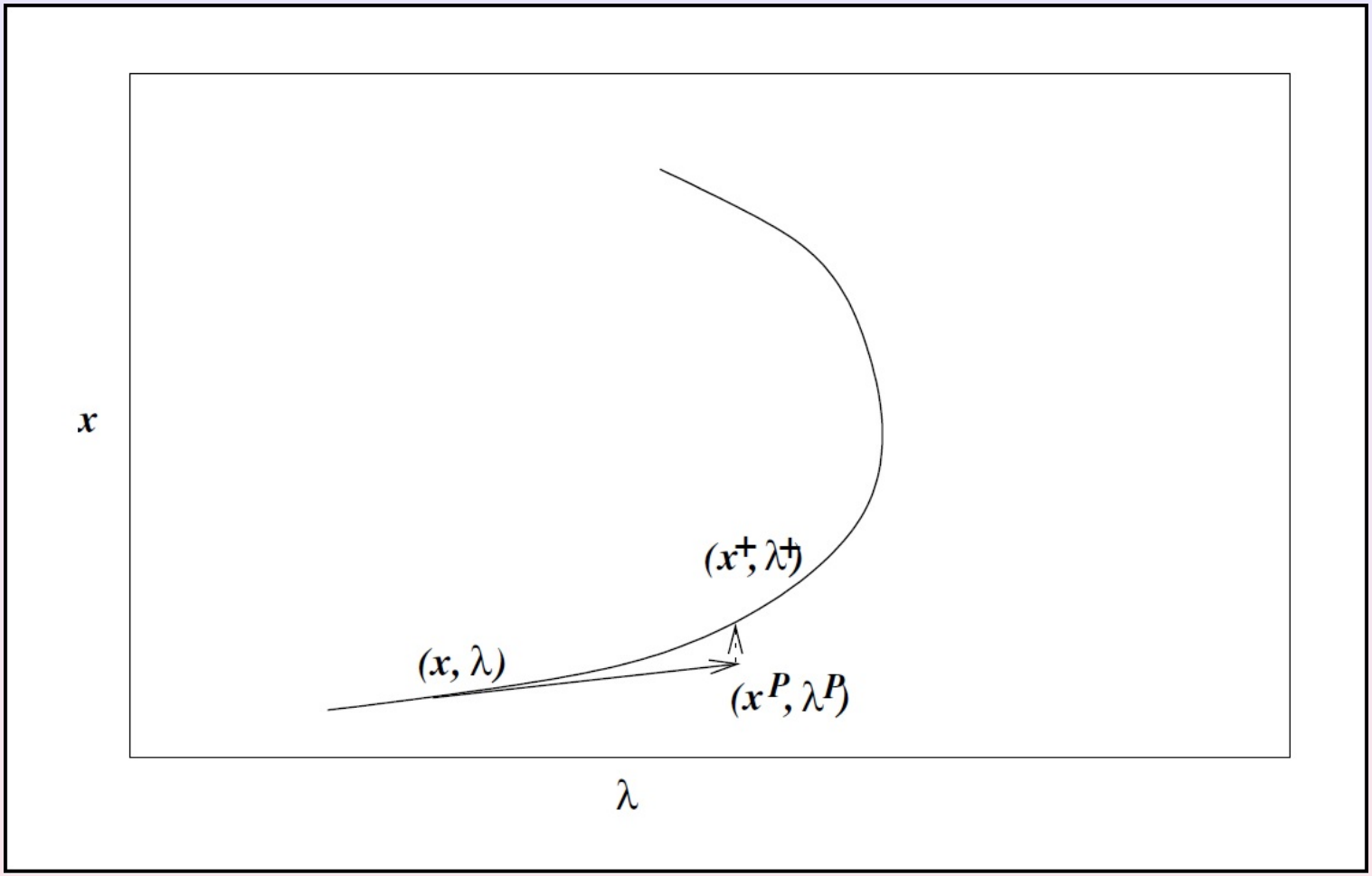

Figure 5: The algebraic predictor-corrector procedure, using λ as the fixed variable in the correction process.

§11.3 Continuation/Homotopy Methods

During the corrections, we choose a component of the predictor step (x^P, λ^P) – one of the components that has been changing most rapidly during the past few steps – and hold this component fixed during the correction process. If the index of this component is j, and if we use a pure Newton corrector process (often adequate, since (x^P, λ^P) is usually quite close to the target point (x^+, λ^+)), the steps will have the form

$$\begin{bmatrix} \frac{\partial H}{\partial x} & \frac{\partial H}{\partial \lambda} \\ e_j^{\mathrm{T}} & \end{bmatrix} \begin{bmatrix} \delta x \\ \delta \lambda \end{bmatrix} = \begin{bmatrix} -H \\ 0 \end{bmatrix},$$

where the quantities $\partial H/\partial x$, $\partial H/\partial \lambda$, and H are evaluated at the latest point of the corrector process.

§11.3 Continuation/Homotopy Methods

The last row of this system serves to fix the j-th component of $(\delta x, \delta\lambda)$ at zero; the vector $e_j \in \mathbb{R}^{n+1}$ is a vector with $n+1$ components containing all zeros, except for a 1 in the location j that corresponds to the fixed component. Note that in Figure 5 the λ component is chosen to be fixed on the current iteration. On the following iteration, it may be more appropriate to choose x as the fixed component, as we reach the turning point in λ.

The two variants on path-following described above are able to follow curves like those depicted in Figure 4 to a solution of the nonlinear system. They rely, however, on the $n \times (n+1)$ matrix in (33) having full rank for all (x, λ) along the path, so that the tangent vector is well-defined. The following result shows that full rank is guaranteed under certain assumptions.

§11.3 Continuation/Homotopy Methods

The last row of this system serves to fix the j-th component of $(\delta x, \delta\lambda)$ at zero; the vector $e_j \in \mathbb{R}^{n+1}$ is a vector with $n+1$ components containing all zeros, except for a 1 in the location j that corresponds to the fixed component. Note that in Figure 5 the λ component is chosen to be fixed on the current iteration. On the following iteration, it may be more appropriate to choose x as the fixed component, as we reach the turning point in λ.

The two variants on path-following described above are able to follow curves like those depicted in Figure 4 to a solution of the nonlinear system. They rely, however, on the $n \times (n+1)$ matrix in (33) having full rank for all (x, λ) along the path, so that the tangent vector is well-defined. The following result shows that full rank is guaranteed under certain assumptions.

§11.3 Continuation/Homotopy Methods

The last row of this system serves to fix the j-th component of $(\delta x, \delta\lambda)$ at zero; the vector $e_j \in \mathbb{R}^{n+1}$ is a vector with $n+1$ components containing all zeros, except for a 1 in the location j that corresponds to the fixed component. Note that in Figure 5 the λ component is chosen to be fixed on the current iteration. On the following iteration, it may be more appropriate to choose x as the fixed component, as we reach the turning point in λ.

The two variants on path-following described above are able to follow curves like those depicted in Figure 4 to a solution of the nonlinear system. They rely, however, on the $n \times (n+1)$ matrix in (33) having full rank for all (x, λ) along the path, so that the tangent vector is well-defined. The following result shows that full rank is guaranteed under certain assumptions.

§11.3 Continuation/Homotopy Methods

Theorem

Suppose that r is twice continuously differentiable. Then for almost all vectors $a \in \mathbb{R}^n$, there is a zero path emanating from $(0, a)$ along which the $n \times (n+1)$ matrix

$$\begin{bmatrix} \frac{\partial}{\partial x} H(x, \lambda) & \frac{\partial}{\partial \lambda} H(x, \lambda) \end{bmatrix} \tag{33}$$

has full rank. If this path is bounded for $\lambda \in [0, 1)$, then it has an accumulation point $(\bar{x}, 1)$ such that $r(\bar{x}) = 0$. Furthermore, if the Jacobian $J(\bar{x})$ is non-singular, the zero path between $(a, 0)$ and $(\bar{x}, 1)$ has finite arc length.

The theorem assures us that unless we are unfortunate in the choice of a, the algorithms described above can be applied to obtain a path that either diverges or else leads to a point $\bar{x}$ that is a solution of the original nonlinear system if $J(\bar{x})$ is non-singular.

§11.3 Continuation/Homotopy Methods

Theorem

Suppose that r is twice continuously differentiable. Then for almost all vectors $a \in \mathbb{R}^n$, there is a zero path emanating from $(0, a)$ along which the $n \times (n+1)$ matrix

$$\left[\begin{array}{cc} \frac{\partial}{\partial x} H(x, \lambda) & \frac{\partial}{\partial \lambda} H(x, \lambda) \end{array} \right] \tag{33}$$

has full rank. If this path is bounded for $\lambda \in [0, 1)$, then it has an accumulation point $(\bar{x}, 1)$ such that $r(\bar{x}) = 0$. Furthermore, if the Jacobian $J(\bar{x})$ is non-singular, the zero path between $(a, 0)$ and $(\bar{x}, 1)$ has finite arc length.

The theorem assures us that unless we are unfortunate in the choice of *a*, the algorithms described above can be applied to obtain a path that either diverges or else leads to a point $\bar{x}$ that is a solution of the original nonlinear system if $J(\bar{x})$ is non-singular.

§11.3 Continuation/Homotopy Methods

We conclude with an example to show that divergence of the zero path – the less desirable outcome of the theorem above – can happen even for innocent-looking problems.

Example

Consider the system $r(x) = x^2 - 1$, for which there are two non-degenerate solutions $+1$ and -1. Suppose we choose $a = -2$ and attempt to apply a continuation method to the function

$$H(x, \lambda) = \lambda(x^2 - 1) + (1 - \lambda)(x + 2) = \lambda x^2 + (1 - \lambda)x + (2 - 3\lambda),$$

obtained by substituting into (31). The zero paths for this function are plotted in Figure 6. As can be seen from that diagram, there is no zero path that joins $(-2, 0)$ to either $(1, 1)$ or $(-1, 1)$, so the continuation methods fail on this example.

§11.3 Continuation/Homotopy Methods

Example (cont'd)

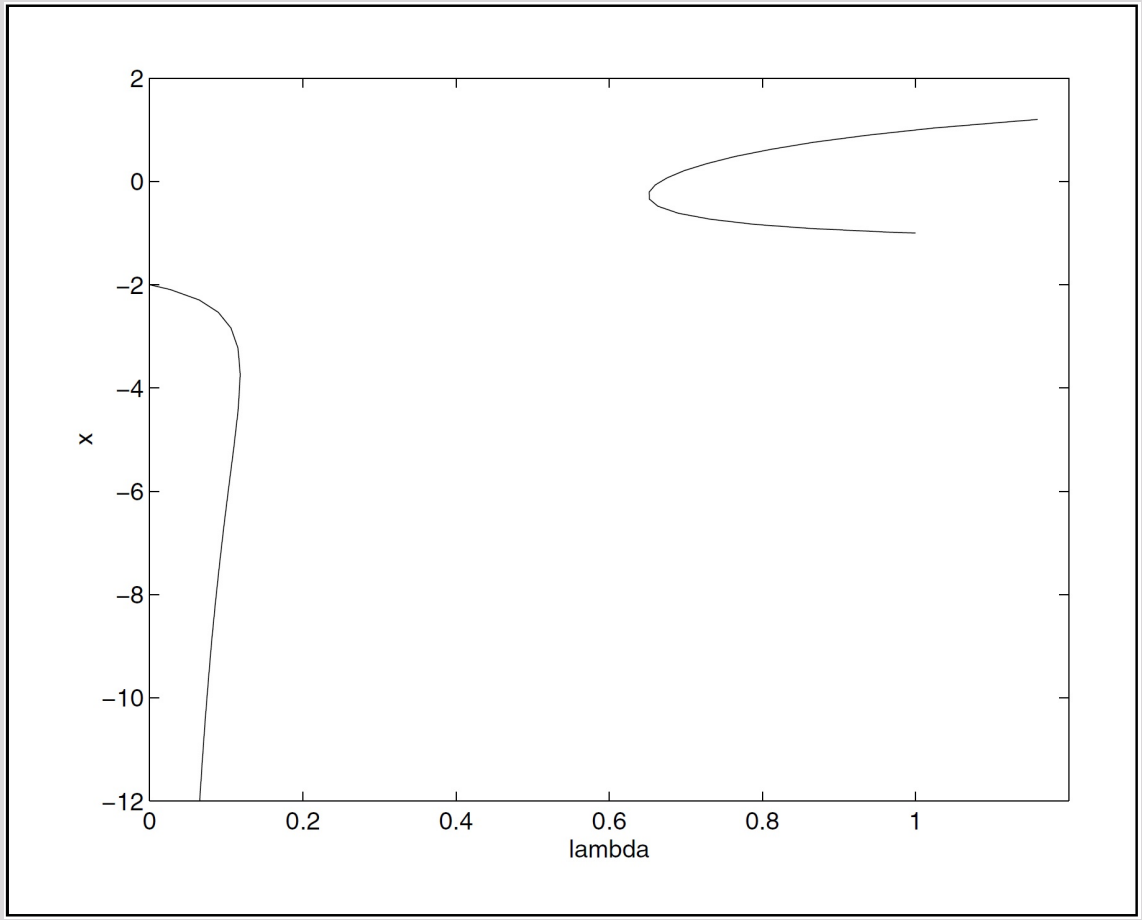

Figure 6: Zero paths for the example. There is no continuous zero path from $\lambda = 0$ to $\lambda = 1$.

§11.3 Continuation/Homotopy Methods

Example (cont'd)

Using the formula for a quadratic root, for a fixed λ the solution to

$$H(x,\lambda) = \lambda x^2 + (1-\lambda)x + (2-3\lambda) = 0$$

is given by

$$x = \frac{-(1-\lambda) \pm \sqrt{(1-\lambda)^2 - 4\lambda(2-3\lambda)}}{2\lambda}\,.$$

Now, when the term in the square root is negative, the corresponding values of x are complex; that is, there are no real roots x. It is easy to verify that such is the case when

$$\lambda \in \left(\frac{5-2\sqrt{3}}{13}, \frac{5+2\sqrt{3}}{13}\right) \approx (0.118, 0.651)\,.$$

Note that the zero path starting from $(-2, 0)$ becomes unbounded, which is one of the possible outcomes of the theorem just stated.

§11.3 Continuation/Homotopy Methods

This example indicates that continuation methods may fail to produce a solution even to a fairly simple system of nonlinear equations. However, it is generally true that the Homotopy methods are more reliable than the merit-function methods described earlier in the chapter. The extra robustness comes at a price, since continuation methods typically require significantly more computational effort than the merit-function methods.