Conventional Newton Methods
## TABLE OF CONTENTS

| §25.1  | Introduction                          | 25–3 |
| §25.2  | Analysis Review                       | 25–3 |
| §25.3  | Problem Statement                     | 25–4 |
| §25.4  | The Corrective Phase                  | 25–4 |
| §25.5  | Solving the Newton Systems            | 25–5 |
| §25.5.1| Newton Iteration Under Load Control   | 25–6 |
| §25.5.2| Newton Iteration Under State Control  | 25–7 |
| §25.5.3| Newton Iteration Under Arclength Control | 25–7 |
| §25.5.4| Newton Iteration Under Hyperspherical Control | 25–7 |
| §25.5.5| Newton Iteration Under Work Control   | 25–8 |
| §25.5.6| Energy Variation in Newton Cycle      | 25–8 |
| §25.6  | Termination Tests                     | 25–8 |
| §25.7  | A Scalar Implementation               | 25–10 |
| §25.8  | A Matrix Implementation               | 25–15 |
| §25.9  | The Ordinary Newton Method            | 25–17 |
| §25.   | Notes and Bibliography                | 25–21 |
| §25.   | Exercises                             | 25–22 |
§25.1. Introduction

In the overview of solution methods given in Chapter 20 it was noted that continuation-based solution methods generally included two phases: incremental and corrective. In the purely incremental methods covered in Chapters 21–24, corrections are omitted. If the corrective phase is present, the incremental formula functions as a predictor that provides a starting point for the corrective iteration. The purpose of this iteration is to eliminate (or at least reduce) the drift error by moving towards the equilibrium path along the increment constraint hypersurface.

Solution methods that include a corrective phase receive several equivalent names in the literature:

- Corrective methods
- Incremental-iterative methods
- Predictor-corrector (PC) methods

The last designation is standard in the numerical solution of ODE by linear multistep methods. It is not emphasized here because there are differences as well as similarities; for example a residual does not necessarily exist in the ODE encountered in real-time dynamics. There are purely corrective methods that lack a predictor phase entirely, which means that iterations start from the previous corrected solution. These have not proven important, however, in practical applications.

The most important class of corrective methods pertains to the Newton-Raphson method and its numerous variants: relaxed, modified, modified-delayed, damped, restricted, quasi, and so forth. This alphabet soup is collectively known as Newton-like methods. Their common feature is that they only require access to the previous solution iterate.1 In the present Chapter we study the conventional Newton method under general increment control.

Remark 25.1. Who was Raphson? Some have (tongue in cheek) suggested that he was Newton’s programmer. For the true story, see [319, p. 64]. Newton did give a version in 1669, in which scalar polynomial equations are solved by an equivalent method involving translation of origin and linearization, but did not explicitly use derivatives (he had not invented that yet). Raphson generalized and presented the method in 1690. In what follows the Raphson name will be dumped.

§25.2. Analysis Review

Let us recall our main goal: to solve the residual equations

\[ r(u, \lambda) = 0, \]  

(25.1)

over a loading stage as the control parameter \( \lambda \) is moved away from 0. (Under general increment control, \( \lambda \) may go up or down.) As discussed in §20.3.1, the additional equation that makes (25.1) determinate is the increment constraint equation, which in generic form reads

\[ c(u, \lambda) = 0. \]  

(25.2)

1 This is noted as a way of contrast with methods that extrapolate from multiple past iterates, called multipoint iteration methods in the literature. Those can be extremely unreliable if not in the vicinity of the solution, and should be avoided.

There is another class of methods that retains higher derivatives in the Taylor expansion of the residual, and which are primarily useful for scalar equations. A compilation of those may be found in [778, Ch. 11].
Chapter 25: CONVENTIONAL NEWTON METHODS

In practice a more adjustable form of (25.2) is used. This is specified in terms of increments from a previous solution as in the second of (25.7) below, which is reset after each incremental step.

Starting from $\lambda = 0$, we want to calculate a series of solutions

$$u_0, \lambda_0, u_1, \lambda_1, \ldots u_n, \lambda_n \ldots$$

(25.3)

that characterizes numerically the response $u = u(\lambda)$ while satisfying the residual equations (25.1) as well as (25.2), within prescribed accuracy. The purely incremental methods covered in Chapters 21–24 compute a sequence of values such as (25.3) by direct numerical integration of the first-order rate equations

$$K \dot{u} = q \dot{\lambda}.$$ 

(25.4)

The methods considered here implement a corrective phase in which one iterates on (25.1) for equilibrium while satisfying the constraint (25.2). The starting point for this phase is the solution predicted by the incremental method.

§25.3. Problem Statement

Assume that $n$ incremental steps of the stage analysis have been performed. The last accepted solution is

$$u_n, \lambda_n.$$ 

(25.5)

Our target is the next solution

$$u_{n+1}, \lambda_{n+1}.$$ 

(25.6)

that satisfies, within specified tolerance, the nonlinear algebraic system

$$r(u_{n+1}, \lambda_{n+1}) = 0, \quad c(\Delta u_n, \Delta \lambda_n) = 0,$$

(25.7)

in which

$$\Delta u_n = u_{n+1} - u_n, \quad \Delta \lambda_n = \lambda_{n+1} - \lambda_n,$$

(25.8)

are increments from the past solution. Instances of the increment constraint are given in §20.5.

The predicted solution is

$$u_{n+1}^0 = u_n + \Delta u_n^0, \quad \lambda_{n+1}^0 = \lambda_n + \Delta \lambda_n^0,$$

(25.9)

is typically obtained by performing an incremental step as described in Chapters 21-24..
§25.4. The Corrective Phase

All computations that follow pertain to the \( (n+1)^{th} \) incremental step. For simplicity we shall omit the step subscript \( n \) from ensuing formulas. Starting from the predicted approximation (25.9),

\[
\begin{align*}
\mathbf{u}^0_{n+1} &\rightarrow \mathbf{u}^0, \\
\lambda^0_{n+1} &\rightarrow \lambda^0,
\end{align*}
\]

(25.10)
a Newton-like method applied to the algebraic system (25.7) generates a sequence of iterates

\[
\begin{align*}
\mathbf{u}^k, \quad \lambda^k,
\end{align*}
\]

(25.11)
in which \( k = 1, 2 \ldots \) is an iteration step index. The conventional Newton method is based on linearizing the governing system (25.7) about the last iterate \((\mathbf{u}^k, \lambda^k)\). This can be accomplished by truncating the control-state Taylor series as

\[
\begin{align*}
\mathbf{r}^{k+1} &= \mathbf{r}^k + \frac{\partial \mathbf{r}}{\partial \mathbf{u}} (\mathbf{u}^{k+1} - \mathbf{u}^k) + \frac{\partial \mathbf{r}}{\partial \lambda} (\lambda^{k+1} - \lambda^k) + H.O. = 0, \\
\mathbf{c}^{k+1} &= \mathbf{c}^k + \frac{\partial \mathbf{c}}{\partial \mathbf{u}} (\mathbf{u}^{k+1} - \mathbf{u}^k) + \frac{\partial \mathbf{c}}{\partial \lambda} (\lambda^{k+1} - \lambda^k) + H.O. = 0.
\end{align*}
\]

(25.12)

Here ‘H.O.’ denote higher order terms that are quadratic or higher in the changes \( \mathbf{u}^{k+1} - \mathbf{u}^k \) and \( \lambda^{k+1} - \lambda^k \). All derivatives are evaluated at \((\mathbf{u}^k, \lambda^k)\). Discarding the H.O. terms and recalling that

\[
\begin{align*}
\mathbf{K} &= \frac{\partial \mathbf{r}}{\partial \mathbf{u}}, \quad \mathbf{q} = -\frac{\partial \mathbf{r}}{\partial \lambda}, \quad \mathbf{a}^T = \frac{\partial \mathbf{c}}{\partial \mathbf{u}}, \quad \mathbf{g} = \frac{\partial \mathbf{c}}{\partial \lambda},
\end{align*}
\]

(25.13)

we find that the corrections

\[
\begin{align*}
\mathbf{d} &= \mathbf{u}^{k+1} - \mathbf{u}^k, \\
\eta &= \lambda^{k+1} - \lambda^k,
\end{align*}
\]

(25.14)
can be obtained by solving the linear algebraic system

\[
\begin{align*}
\begin{bmatrix}
\mathbf{K} & -\mathbf{q} \\
\mathbf{a}^T & \mathbf{g}
\end{bmatrix}
\begin{bmatrix}
\mathbf{d} \\
\eta
\end{bmatrix}
= -
\begin{bmatrix}
\mathbf{r} \\
\mathbf{c}
\end{bmatrix}.
\end{align*}
\]

(25.15)

Here all known quantities are evaluated at \( \mathbf{u}^k \) and \( \lambda^k \), for example \( \mathbf{r}^k = \mathbf{r}(\mathbf{u}^k, \lambda^k) \). For notational simplicity, however, the \( k \) superscript will kept out of \( \mathbf{d}, \eta, \mathbf{r}, \mathbf{K}, \mathbf{q}, \mathbf{a}, \) and \( \mathbf{g} \), unless it is desirable to make the dependency on the iteration index explicit. If the tangent stiffness matrix \( \mathbf{K} \) is of order \( N \), the coefficient matrix of the linear system (25.15) has order \( N + 1 \). This matrix is called the augmented stiffness matrix.

Note that although generally the tangent stiffness \( \mathbf{K} \) is symmetric and sparse, the augmented stiffness is generally unsymmetric (but see Exercise 25.2). Furthermore, its sparseness may be detrimentally affected by the augmentation if the row vector \( \mathbf{a}^T \) links all DOF. It is therefore of interest to treat the linear system (25.15) with techniques that preserve both symmetry and sparseness.

The solution procedures described below make use of auxiliary systems of equations to achieve that goal. The number of auxiliary systems depends on whether the tangent stiffness \( \mathbf{K} \) is nonsingular (regular points) or singular (critical points). For the latter we have to distinguish between limit points and bifurcation points. In the present Chapter we shall concentrate on the treatment of regular points.
Table 25.1. Incremental Control Strategies for Newton Solver — Unscaled

<table>
<thead>
<tr>
<th>Term</th>
<th>Acronym</th>
<th>Constraint $c$</th>
<th>$a$</th>
<th>$g$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Control</td>
<td>LC</td>
<td>$\Delta \lambda - \ell = 0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>State Control</td>
<td>SC</td>
<td>$</td>
<td></td>
<td>\Delta u^T \Delta u - \ell^2 = 0$</td>
</tr>
<tr>
<td>Modified State Control</td>
<td>MSC</td>
<td>$\sqrt{\Delta u^T \Delta u - \ell} = 0$</td>
<td>$2\Delta u/(\Delta u^T \Delta u)$</td>
<td>0</td>
</tr>
<tr>
<td>Arc-length Control</td>
<td>AC</td>
<td>$(\Delta u - \ell)/f = 0$</td>
<td>$v_g/s/f_s$</td>
<td>$1/f_s$</td>
</tr>
<tr>
<td>Hyperspherical Control</td>
<td>HC</td>
<td>$\Delta u^T \Delta u + \Delta \lambda^2 - \ell = 0$</td>
<td>$2\Delta u$</td>
<td>$2\Delta \lambda$</td>
</tr>
<tr>
<td>Modified Hyperspherical</td>
<td>MHC</td>
<td>$\sqrt{</td>
<td></td>
<td>\Delta u^T \Delta u + \Delta \lambda^2 - \ell} = 0$</td>
</tr>
<tr>
<td>External Work Control</td>
<td>EWC</td>
<td>$-q^T \Delta u + \ell = 0$</td>
<td>$-q$</td>
<td>0</td>
</tr>
</tbody>
</table>

§25.5. Solving the Newton Systems

Recall from Chapter 4 that regular points of (25.1) are equilibrium solutions $(\mathbf{u}, \lambda)$ at which the tangent stiffness matrix $\mathbf{K} = \partial \mathbf{r}/\partial \mathbf{u}$ is nonsingular. If this property holds at the current solution iterate, we can perform forward Gauss elimination on (25.15) to get rid of $\mathbf{d}$ and produce the following scalar equation for $\eta$:  

\[
(g + a^T K^{-1} q) \eta = -c + a^T K^{-1} r.  \tag{25.16}
\]

Let $\mathbf{v}_r$ and $\mathbf{v}_q$ denote the solution of the symmetric linear systems

\[
\begin{align*}
\mathbf{Kv}_r &= -\mathbf{r}, \\
\mathbf{Kv}_q &= \mathbf{q}.
\end{align*} \tag{25.17}
\]

Here $\mathbf{v}_q$ is our old friend the incremental velocity vector $\mathbf{v} = K^{-1}$, whereas $\mathbf{v}_r = -K^{-1}\mathbf{r}$ is called the residual velocity vector. (Subscripts are appended to distinguish the right-hand sides). Using (25.17) and solving for $\eta$ from (25.16), the solution of (25.15) can be compactly expressed as

\[
\eta = -\frac{c + a^T v_r}{g + a^T v_q}, \quad \mathbf{d} = \mathbf{v}_r + \eta \mathbf{v}_q. \tag{25.18}
\]

It is seen that two right hand sides, $\mathbf{r}$ and $\mathbf{q}$, have to be generally solved for at each Newton step. The number reduces to one for $k > 1$, however, if (i) modified Newton is used so that $\mathbf{K}$ is held fixed for several steps and (ii) $\mathbf{q}$ does not vary. The second assumption holds in structural mechanics applications if the loading is conservative and proportional. (The modified Newton method is described in Chapter 26.)

Three specializations are illustrated below. In all instances the incremental step number $n$ and the iteration index $k$ are restored for clarity.

\[\text{Another way to get (25.16) is to solve for } \mathbf{d} \text{ from the first equation: } \mathbf{d} = K^{-1}(q \eta - \mathbf{r}), \text{ and replace } \mathbf{d} \text{ into the second one: } a^T \mathbf{d} + g \eta = -c.\]
§25.5 SOLVING THE NEWTON SYSTEMS

25.5.1. Newton Iteration Under Load Control

For load control, also called $\lambda$ control (an after name for non-structural applications), $c = \Delta \lambda_n - \ell_n = 0$, whence $a = 0$ and $g = 1$. Thus

$$\eta^k = -c^k, \quad d^k = v^k_r - c^k v^k_q.$$  \hspace{1cm} (25.19)

If all solution iterates satisfy the constraint $c = 0$, then $\eta^k = 0$ and $d^k = v^k_r = -K^{-1} r^k$. If so the corrective process reduces to the well known form of the conventional Newton iteration:

$$\lambda_{n+1}^k = \lambda_n^k = \lambda_n - \ell_n, \quad u_{n+1}^k = u_n^k + v^k_r = u_n^k - (K^{-1})^{-1} r^k.$$  \hspace{1cm} (25.20)

In this case $\lambda_{n+1}^k$ stays fixed for all $k$, and only the solution $v^k_r$ for the residual RHS is needed. The traditional presentation of Newton’s method in the literature focuses on this particular case.

25.5.2. Newton Iteration Under State Control

For unscaled state control $c = c(u_n, \lambda_n) = \Delta u_n^T \Delta u_n - \ell_n^2 = 0$, whence $a = 2 \Delta u_n$ and $g = 0$. The Newton iteration becomes

$$\eta^k = -c^k + 2(\Delta u_n^k)^T v^k_q / 2(\Delta u_n^k)^T v^k_q, \quad d^k = v^k_r + \eta^k v^k_q, \quad \lambda^{k+1} = \lambda^k + \eta^k, \quad u^{k+1} = u^k + d^k.$$  \hspace{1cm} (25.21)

The formula for $\eta^k$ simplifies slightly if all iterates satisfy the constraint, in which case $c^k = 0$. For scaled state control: $c = (\Delta u_n^T / L_{ref})^2 - \ell_n^2 = 0$, in which $L_{ref}$ is a constant reference length, the modifications are obvious.

Note that if $\Delta u_n \to 0$ in a state space region, $a = 2 \Delta u_n \to 0$, whence the last row of the augmented stiffness vanishes (because $g = 0$) and the iteration fails even if $K$ is nonsingular. This may occur in the vicinity of turning points. To alleviate this problem the constraint may be changed to

$$c(u_n, \lambda) = ||\Delta u_n|| - \ell_n = \sqrt{\Delta u_n^T \Delta u_n} - \ell_n = 0.$$  \hspace{1cm} (25.22)

If so, $a = \partial c / \partial u = 2 \Delta u / ||\Delta u||$. If $||\Delta u|| \to 0$, the $a$ components go to 0/0, which is less deadly, and may approach nonzero limits depending on component ratios. Singularity may still be lurking but can be avoided with simple perturbations in $\Delta u$. For a scalar equation as in the example below the modified state constraint is simply $u - \ell_n = 0$, which is safe since $a = \partial c / \partial u = 1$.

25.5.3. Newton Iteration Under Arclength Control

For unscaled arclength control the constraint $c = 0$ is given as (20.27) of Chapter 20. In this case $a = v_n / f_n$ and $g = 1 / f_n$, in which $f_n = \sqrt{1 + v_n^T v_n}$, giving

$$\eta^k = -c^k + (1/f_n) (v^k_n)^T v^k_q / (1/f_n) (1 + (v^k_n)^T v^k_q), \quad d^k = v^k_r + \eta^k v^k_q, \quad \lambda^{k+1} = \lambda^k + \eta^k, \quad u^{k+1} = u^k + d^k.$$  \hspace{1cm} (25.23)

If all solution iterates satisfy the constraint $c = 0$, the factor $1/f_n$ in $\eta^k$ cancels out.

For scaled arclength control, in which $v^k_n$ is replaced by $\tilde{v}^k_n$ some minor modifications have to be made.

---

3 Applied mathematicians working in optimization sometimes call it horizontal Newton – a rather funny name.
§25.5.4. Newton Iteration Under Hyperspherical Control

For unscaled hyperspherical control the constraint is
\[ c(\Delta u, \Delta \lambda) = (\Delta u)^2 + \Delta \lambda^2 - \ell_n = 0, \]  
(25.24)
In this case \( \mathbf{a} = 2\Delta \mathbf{u} \) and \( g = 2\Delta \lambda \). The iteration is
\[ \eta^k = -c^k + (1/f_n)(\mathbf{v}_n^k)^T \mathbf{v}_q^k \quad \frac{(1/f_n)(1 + (\mathbf{v}_n^k)^T \mathbf{v}_q^k)}{\mathbf{d}^k = \mathbf{v}_r^k + \eta^k \mathbf{v}_q^k, \quad \lambda^{k+1} = \lambda^k + \eta^k, \quad \mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{d}^k}. \]  
(25.25)

If all solution iterates satisfy the constraint \( c = 0 \), the factor \( 1/f_n \) in \( \eta^k \) cancels out.

For scaled hyperspherical control, in which \( \mathbf{v}_n^k \) is replaced by \( \mathbf{v}_r^k \) some minor modifications have to be made.

§25.5.5. Newton Iteration Under Work Control

For unscaled work control the constraint is
\[ c(\Delta u, \Delta \lambda) = -\mathbf{q}^T \Delta \mathbf{u} + \ell_n = 0, \]  
(25.26)
(The minus sign is chosen for \( \mathbf{q} \) so that the augmented stiffness becomes symmetric.) In this case \( \mathbf{a}^T = -\mathbf{q}^T \) and \( g = 0 \). The iteration is
\[ \eta^k = -c^k + (1/f_n)(\mathbf{v}_n^k)^T \mathbf{v}_q^k \quad \frac{\mathbf{d}^k = \mathbf{v}_r^k + \eta^k \mathbf{v}_q^k, \quad \lambda^{k+1} = \lambda^k + \eta^k, \quad \mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{d}^k}{\eta^k}}. \]  
(25.27)

Work control does not need scaling, but blows up if \( \mathbf{q} \) vanishes.

§25.5.6. Energy Variation in Newton Cycle

Assume the system is conservative, with total potential energy (TPE) of the form \( \Pi(u, \lambda) = U(u) - W(u, \lambda) \). Consider the TPE variation over the \( k^{th} \) iteration cycle. The starting state and control variables are \( u^k \) and \( \lambda^k \) and the TPE is \( \Pi^k = \Pi(u^k, \lambda^k) \). Upon cycle completion the TPE is \( \Pi^{k+1} = \Pi(u^{k+1}, \lambda^{k+1}) \). The change \( \Delta \Pi^k = \Pi^{k+1} - \Pi^k \) can be Taylor-expanded up to second order in \( \mathbf{d}^k \) and \( \eta^k \) to get
\[ \Delta \Pi^k = \left[ (\mathbf{d}^k)^T \right] \left[ \begin{array}{c} \frac{\partial \Pi}{\partial \mathbf{u}} \\ \frac{\partial \Pi}{\partial \lambda} \end{array} \right] \left[ \begin{array}{c} \mathbf{d}^k \\ \eta^k \end{array} \right] + \frac{1}{2} \left[ (\mathbf{d}^k)^T \right] \left[ \begin{array}{c} \frac{\partial^2 \Pi}{\partial \mathbf{u}^2} \\ \frac{\partial^2 \Pi}{\partial \lambda^2} \end{array} \right] \left[ \begin{array}{c} \mathbf{d}^k \\ \eta^k \end{array} \right] \]  
(25.28)

in which \( f^k = (\partial \Pi/\partial \lambda)^k \) and \( b^k = (\partial^2 \Pi/\partial \lambda^2)^k \). Next, selectively replace \( \mathbf{d}^k = \mathbf{v}_r^k + \eta^k \mathbf{v}_q^k = (K^{-1})^k (-\mathbf{r}^k + \eta^k \mathbf{q}^k) \) to get rid of all occurrences of \( K \), leaving only vectors dot products:
\[ \Delta \Pi^k = \frac{1}{2} (\mathbf{r}^k)^T \mathbf{d}^k + \eta^k \left[ f^k + \frac{1}{2}(\mathbf{v}_q^k)^T \mathbf{r}^k \right] + \mathbf{d}^k \]  
(25.29)

If the contribution of \( \eta^k \) is neglected, this simplifies to
\[ \Delta \Pi^k = \frac{1}{2} (\mathbf{r}^k)^T \mathbf{d}^k. \]  
(25.30)

This expression is correct for load control once the Newton iterates move along the \( \lambda \)-constant surface, but it can be significantly in error for other strategies.
§25.6. Termination Tests

At which point should we stop the Newton iteration? Several convergence criteria that can be applied. Three are described below.

**Displacement convergence test.** The change in the last state correction $d^k$ as measured in an appropriate norm, should be less than a given value. For example, using the 2-norm (Euclidean norm)

$$
||d||_2 = \sqrt{d^T d} \leq \epsilon_d ||d_{ref}||_2.
$$

(25.31)

in which $d_{ref}$ is a reference displacement and $\epsilon_d$ is a tolerance such as $10^{-6}$. A common way to chose the reference is to pick $d_{ref} = d^0$, the displacement correction after the first cycle. An alternative is to pick a a threshold value $d_{tol}$ in the RHS.

One problem with this convergence measure is that displacement DOF do not necessarily have the same physical dimension. This occurs when both rotational and translational DOF are present. Another possibility is that $u$ includes lagrange multipliers that have dimensions of force. In these scenarios the introduction of a diagonal scaling matrix $S$ is recommended so that $d^T d$ is replaced by $d^T S d$.

**Residual convergence test.** Since the residual $r$ measures the departure from equilibrium, a convergence test formally similar to (25.31) is

$$
||r||_2 = \sqrt{r^T r} \leq \epsilon_r ||r_{ref}||_2.
$$

(25.32)

in which $r_{ref}$ is a reference residual and $\epsilon_r$ is a tolerance such as $10^{-6}$. A common way to chose the reference is to pick $r_{ref} = r^1$, the residual after the first cycle. The same warnings noted above regarding different physical dimensions apply.

**Energy convergence test.** This approach combines the residual and displacement-change vectors. The linearized internal energy change in the $k^{th}$ cycle is

$$
\Delta W^k = \frac{1}{2} d^k K^k d^k = \frac{1}{2} (d^k)^T (\eta^k q^k - r^k).
$$

(25.33)

in which the last equality is the matter of an Exercise. Since $\Delta W^k$ is quadratic in $d^k$, it is convenient to take the (positive) square root and to set up the convergence test as

$$
\sqrt{\Delta W^k} < \epsilon_W \sqrt{\Delta W_{ref}},
$$

(25.34)

in which $\Delta W_{ref}$ is a reference energy change and $\epsilon_W$ is a numerical tolerance such as $10^{-6}$. The value of $\Delta W_{ref}$ may be selected in various ways; for example in a predictor-corrector scheme it may be the energy change in the predictor step. Or it may be supplied by the calling program.

The great advantage of the energy test is that no scaling is needed. Physical unit differences in state DOF are exactly compensated by the conjugate forces in $r$ and $q$.

**Divergence Safeguards.** The Newton iteration is not guaranteed to converge. There should therefore be divergence detection tests that will cause the iteration to be interrupted before the maximum
Figure 25.1. A module that carries out Newton iteration on a scalar residual function \( r(u, \lambda) \).

Several increment control strategies implemented. Used in Examples 25.1 through 25.7.

The number of cycles is reached. For example, turning the above ratios around, divergence may be diagnosed if either of the following inequalities occur:

\[
\frac{\|r\|}{\|r^0\|} \geq g_r, \quad \frac{\|d\|}{\|u^0\|} \geq g_d
\]  

(25.35)

where \( g_r \) and \( g_d \) are “dangerous growth” factors, for example \( g_r = g_d = 100 \).

Often the Newton iteration will neither diverge nor converge but just “bounce around” (oscillatory behavior). To avoid excessive wheel spinning in such cases it is always a good practice to put a maximum number of iterations per step in the program. A typical number may be 10.

§25.7. A Scalar Implementation

The implementation listed in Figure 25.1 carries out the Newton iteration process on a scalar residual function \( r(u, \lambda) \), assumed to be separable: \( r(u, \lambda) = p(u) - f(\lambda) \). It is used for ensuing examples. Several increment control strategies may be specified through an argument to facilitate performance comparison. The module is invoked as

\[
\{u\lambda C, ec\} = \text{ScalarNewton}[r, K, q, u\lambda 0, u\lambda P, cspar, extpar, modpar, prtab]
\]  

(25.36)
The arguments are
\[ r,K,q \]
Three externally defined functions that return the residual vector \( r \), the tangent stiffness matrix \( K = \partial r / \partial u \) and the incremental load vector \( q = -\partial r / \partial \lambda \), respectively. The residual has two arguments: \( u \) and \( \lambda \); the stiffness has one argument: \( u \), and the incremental load has one argument: \( \lambda \).

\[ u\lambda S \]
A 2-item list: \( \{ uS, \lambda S \} \) called the reference state-control pair. The increments are defined as \( \Delta u^k = u^k - u_S \) and \( \Delta \lambda^k = \lambda^k - \lambda_S \). (In an actual incremental analysis, this will be the previous solution, but here it need not be.)

\[ u\lambda P \]
A 2-item list: \( \{ uP, \lambda P \} \) called the predicted state-control pair, from which the Newton iteration starts: \( u^0 = u_P \) and \( \lambda^0 = \lambda_P \). Need not satisfy the increment constraint and may in fact coincide with \( \{ uS, \lambda S \} \).

\[ cspar \]
A two item list: \( \{ ics, ell \} \) that specifies the increment control strategy:

\[ ics \]
A character string specifying the increment control strategy to be used; e.g., "AC" for arclength control. For others, see Table ?. If none of those, "LC" is assumed.

\[ ell \]
The increment length \( \ell \) that appears in the constraint equation.

\[ extpar \]
A list of 3 values: \( \{ kmax, dtol, rtol \} \) that regulates exit from the Newton iteration. \( kmax \) is maximum number of iteration cycles. \( dtol \) is a relative-change tolerance to exit if \( |d^k| < dtol \ell \). \( rtol \) is a residual norm tolerance to exit if \( |r^k| < rtol |r_P| \), in which \( r_P \) is the residual evaluated at \( \{ uP, \lambda P \} \).

\[ modpar \]
A list of modification parameters. Presently only one: \( \{ omega \} \), in which \( \omega \) is a step-relaxation parameter discussed in the next chapter. Normally set to 1.

\[ protab \]
An integer that specified whether the module is to build and print a solution table that records the progress of the iteration. If zero, this task is omitted. If \( n = prtab > 0 \), \( n \) is used as a guide for the number of decimal places recorded in the table.

The function returns \( \{ u\lambda C, ec \} \), in which

\[ u\lambda C \]
The exit state-control pair \( \{ uC, \lambda C \} \) (here “C” is for “corrected” values).

\[ ec \]
Integer error code. If \( ec=k>0 \), convergence was achieved after \( k \) cycles. A \( k=0 \) return indicates no convergence or divergence after \( kmax \) cycles. Negative values flag fatal errors: \( k=-1 \): iteration diverges; \( k=-2 \): stiffness appears singular.

The scalar residual used in the following examples is
\[ r(u, \lambda) = 4u - 4u^2 - \lambda, \quad (25.37) \]

The equilibrium path and nearby incremental flow of (25.37) are plotted in Figure 25.2(a). Of particular interest are the equilibrium points
\[ u = 1/4, \lambda = 3/4, \quad u = 1/2, \lambda = 1/2, \quad (25.38) \]

which are labeled \( Q \) and \( L \), respectively, in Figure 25.2(a). The latter is the only limit point.
Example 25.1. Use the ScalarNewton module of Figure 25.1 to find the two equilibrium solutions (25.38) of (25.37) with load control (LC). For both solutions the reference state-control pair is \((u_S = 0, \lambda_S = 0)\). The iterations are also started from that point. The only difference is that \(\ell = 3/4\) for Q and \(\ell = 1\) for L. The driver script is shown in the top cell of Figure 25.3, with output in the bottom cell.

If \(\ell = 3/4\), convergence to Q is rapid and the solution \(u = 1/4\) is captured to full (double precision) accuracy in 5 cycles. The behavior changes for the limit point L targeted by \(\ell = 1\). The results display a slower linear convergence: after 8 cycles only 3 correct digits are obtained. This is typical for a root of multiplicity 2 — note that the exact solution of \(r(u, 1) = 4u - 4u^2 - 1 = 0\) is the double root \(u = 1/2\). Quadratic convergence can be restored by setting the relaxation factor \(\omega\) to 2, but if that is done here the iteration fails with an exactly singular \(K\) after 1 cycle. Instead \(\omega\) is set to 1.99. The limit point is then located to full accuracy after 4 cycles.

If \(\ell > 1\), the process becomes chaotic since the roots become complex, and the iteration does not converge.

Example 25.2.

Example 25.1 is repeated using state control (SC). The constraint is \(\Delta u^2 = \ell^2\). The reference solution is again \((u_S = 0, \lambda_S = 0)\) so we set \(\ell = 1/4\) and \(\ell = 1/2\) to find Q and L, respectively. But starting at \((u_P = 0, \lambda_P = 0)\) fails because of exact singularity after one step. Instead we start from \(u_P = 1/8, \lambda_P = 0\) for both targets.

The driver script is shown in the top cell of Figure 25.4, and the output is listed in the bottom cell. Convergence to full accuracy is observed after 5 and 6 cycles for Q and L, respectively. No linear convergence slowdown is noted for \(\ell = 1/2\); the reason is that the constraint line intersects the response curve at a 90° angle.

It should be noted that use of state control for a simple scalar equation such as (25.37) makes little practical sense since it is trivial to get \(\lambda\) from \(r(u, \lambda) = 0\) given \(u\). It is used here only to compare its performance with other control strategies.

Example 25.3. The previous example is redone with modified state control (MSC). For a scalar equation this is simply \(\Delta u = \ell\) instead of \(\Delta u^2 = \ell^2\). The reference and starting points are the same. The driver script is shown in the top cell of Figure 25.5, and the output is listed in the bottom cell.

Convergence to Q is quick: full accuracy is reached in 2 cycles. As for L, the iteration aborts after one cycle after encountering a stiffness singularity — the same singularity actually occurs regardless of the starting point.
Example 25.4. The example is now carried out using arclength control (AC). The reference solution is again \((u_S = 0, \lambda_S = 0)\). The starting points are \((u_P = 0, \lambda_P = 0)\) for \(Q\) and \((u_P = 1/8, \lambda_P = 0)\) for \(L\). The arclength determination involves some geometric computations, which are illustrated in Figure 25.2(b).

25–13
Because the intersection of the normal line with the response parabola at $L$ occurs at a fairly shallow angle to $Q$ is reasonably fast and full accuracy is reached in 5 cycles. Convergence to $L$ is slowed down, however.

The driver script is shown in the top cell of Figure 25.6, and the output is listed in the bottom cell. Convergence points are $Q$ and $L$ are

Example 25.5

Basically, we want normals to the tangent at $S$ to pass through the target points. That gives the steplengths

$$\ell_Q = 13/(4\sqrt{17}) \quad \ell_L = 9/(2\sqrt{17})$$

The driver script is shown in the top cell of Figure 25.6, and the output is listed in the bottom cell. Convergence to $Q$ is reasonably fast and full accuracy is reached in 5 cycles. Convergence to $L$ is slowed down, however, because the intersection of the normal line with the response parabola at $L$ occurs at a fairly shallow angle—there are in fact two close intersection points but they are difficult to discern in the figure. The iteration eventually picks up quadratic-convergence speed and achieves full accuracy in 8 cycles.

Example 25.5

Next in line to try is hyperspherical control (HC). For a scalar equation this is $(\Delta u)^2 + (\Delta \lambda)^2 = \ell^2$, which is a circle of radius $\ell$. The reference solution is again $(u_S = 0, \lambda_S = 0)$, from which the distances to points $Q$ and $L$ are $\ell = \sqrt{5}/8$ and $\ell = \sqrt{5}/4$, respectively. See Figure 25.2(c) for the geometry. The starting points are $(u_p = 1/4, \lambda_p = 1/2)$.

The driver script is shown in the top cell of Figure 25.7, with the output in the bottom cell. Convergence to $Q$ is reasonably fast and full accuracy is achieved after 5 cycles. Convergence to the limit point is not achieved: the iteration actually bypasses it and zeros in the second intersection point of the parabola and the circle;
\begin{verbatim}
ClearAll[r, K, u, q, \lambda]; itexit = {8, 10.^(-6), 10.^(-6)};
{r[u_, \lambda_] := N[4*u - 4*u^2 - \lambda]; K[u_] := N[4 - 8*u]; q[\lambda_] := 1.};
{uk, k, ec} = ScalarNewton[r, K, q, {0, 0}, {1/4, 1/2}, Sqrt[5/8], 1, itexit, "HC"];
Print["   Exit u = ", uk // InputForm, ", \lambda = ", \lambda k // InputForm, ", ec = ", ec];
{uk, \lambda k, ec} = ScalarNewton[r, K, q, {0, 0}, {1/4, 1/2}, Sqrt[5/4], 1, itexit, "HC"];
Print["   Exit u = ", uk // InputForm, ", \lambda = ", \lambda k // InputForm, ", ec = ", ec];

t
\begin{verbatim}
Exit u = 0.25, \lambda = 0.75, ec = 5
\end{verbatim}

t
\begin{verbatim}
Exit u = 0.6506762326757285, \lambda = 0.9091866916265996, ec = 6
\end{verbatim}
\end{verbatim}

\textbf{Figure 25.7.} Top cell: script that calls the \texttt{ScalarNewton} module of Figure 25.1 for residual (25.37) using the hyperspherical control (HC) strategy. Bottom cell: output results for two cases discussed in the text.

\begin{verbatim}
ClearAll[r, K, u, q, \lambda]; itexit = {8, 10.^(-6), 10.^(-6)};
{r[u_, \lambda_] := N[4*u - 4*u^2 - \lambda]; K[u_] := N[4 - 8*u]; q[\lambda_] := 1.};
{uk, k, ec} = ScalarNewton[r, K, q, {0, 0}, {1/4, 1/2}, Sqrt[5/8], 1, itexit, "MHC"];
Print["   Exit u = ", uk // InputForm, ", \lambda = ", \lambda k // InputForm, ", ec = ", ec];
{uk, \lambda k, ec} = ScalarNewton[r, K, q, {0, 0}, {1/4, 1/2}, Sqrt[5/4], 1, itexit, "MHC"];
Print["   Exit u = ", uk // InputForm, ", \lambda = ", \lambda k // InputForm, ", ec = ", ec];

t
\begin{verbatim}
Exit u = 0.25, \lambda = 0.75, ec = 5
\end{verbatim}

t
\begin{verbatim}
Exit u = 0.49999999999996914, \lambda = 1.0000000000000187, ec = 6
\end{verbatim}
\end{verbatim}

\textbf{Figure 25.8.} Top cell: script that calls the \texttt{ScalarNewton} module of Figure 25.1 for residual (25.37) using the modified hyperspherical control (MHC) strategy. Bottom cell: output results for two cases discussed in the text.

cf. Figure 25.2(c). This is always a possibility when \( r = 0 \) and the circle \( c = 0 \) meet at a shallow angle.

\textbf{Example 25.6.} The previous example is redone with modified hyperspherical control (MSC). For a scalar equation the constraint is \( \sqrt{\Delta u^2 + \Delta \lambda^2} = \ell \).

The driver script is shown in the top cell of Figure 25.6, and the output is listed in the bottom cell. Convergence to point Q is reasonably fast and full precision is reached in 5 cycles. Convergence to L is slower, because the intersection of the circle with the equilibrium at L occurs at a fairly shallow angle. It eventually picks up speed after 3 cycles, with full accuracy achieved after 6 cycles.

\textbf{Example 25.7.} A final test is done with external work control (EWC). For a scalar equation this is \( q \Delta u = \ell \); since \( q = 1 \), this is identical to modified state control (MSC) except for the sign of \( \alpha^k \). The driver script is shown in the top cell of Figure 25.9, and the output is listed in the bottom cell. Convergence to full accuracy is obtained after 2 cycles.
ClearAll[r,K,u,q,λ]; itexit={8,10.^(-6),10.^(-6)};

r[u_,λ_]:=N[4*u-4*u^2-λ];
K[u_]:=N[4-8*u];
q[λ_]:=1.;

{uk,λk,ec}=ScalarNewton[r,K,q,{0,0},{0,0},1/4,1,itexit,"EWC"];
Print["   Exit u=",uk//InputForm,"   λ=",λk//InputForm,"   ec=",ec];

{uk,λk,ec}=ScalarNewton[r,K,q,{0,0},{1/8,0},1/2,1,itexit,"EWC"];
Print["   Exit u=",uk//InputForm,"   λ=",λk//InputForm,"   ec=",ec];

k     rk        Kk       ck      ηk        dk       λkp1       ukp1
0  0.000000  4.00000  0.25000  1.00000  0.2500000  1.000000  0.2500000
1 -0.250000  2.00000  0.00000 -0.250000  0.0000000  0.750000  0.2500000

Exit u=0.25, λ=0.75, ec=2

k     rk        Kk       ck      ηk        dk       λkp1       ukp1
0  0.437500  3.00000  0.37500  1.062500  0.3750000  1.562500  0.5000000

Exit u=0.49999999999999994, λ=1.5625, ec=-2

Figure 25.9. Top cell: script that calls the ScalarNewton module of Figure 25.1 for residual (25.37) using the external work control (EWC) strategy. Bottom cell: output results for two cases discussed in the text.

§25.8. A Matrix Implementation

The implementation listed in Figure 25.10 is more ambitious. It does the Newton iteration process on a vector residual function \( r(u, λ) \) with a small number of state variables. It is intended to be used for simple (non-FEM) examples, and to provide a framework for the actual corrector module implemented in the finite element program GeNoBe. The residual is assumed to be separable: \( r(u, λ) = p(u) - q(λ) \). For the calculation of external work it will be assumed that \( W \approx λu^T\bar{q} \), where \( \bar{q} \) is a mean value.

As in the case of the scalar implementation discussed in §25.7, several increment control strategies may be specified to facilitate performance comparison. The module is invoked as

\[
\{uλC,ec\}=\text{MatrixNewton}\left[r,K,q,uλS,uλP,cspar,expar,accpar,prsol\right]
\]  

The arguments are

- \( r, K, q \): Three externally defined functions that return the residual vector \( r \), the tangent stiffness matrix \( K = \partial r/\partial u \) and the incremental load vector \( qbold = -\partial r/\partial λ \), respectively. The residual has two arguments: \( u \) and \( λ \); the stiffness has one argument: \( u \), and the incremental load has one argument: \( λ \).

- \( uλS \): A 2-item list: \( \{uS,λS\} \) called the reference state-control pair. The increments are defined as \( Δu^k = u^k - uS \) and \( Δλ^k = λ^k - λS \). In an actual incremental analysis, this will be the previous solution, but here it need not be.

- \( uλP \): A 2-item list: \( \{uP,λP\} \) called the predicted state-control pair, from which the Newton iteration starts: \( u^0 = uP \) and \( λ^0 = λP \). Need not satisfy the increment constraint and may in fact coincide with \( uλS \).

- \( cspar \): A two item list: \( \{ics,ell\} \) that defines the increment control strategy:
  - \( ics \) is a character string specifying the increment control strategy to be used; for example "AC" for arclength control. For others, see Table ?. If none of those, "LC" is assumed.
  - \( ell \) is the steplength that appears in the constraint equation.

- \( exitpar \): Exit parameters that regulate the termination of the iteration process. Organized as \( \{kmax,\{dcon,ddiv,dref\},\{rcon,rdiv,rref\},\{Wcon,Wdiv,Wref\},Ksing\} \). Any of the 3 sublists may be empty to skip one or more tests as described below.

25–16
MatrixNewton[r_, K_, q_, u[S], u[P], cspar_, exitpar_, modpar_, histab_] :=
Module[{uS, λS, rS, KS, qS, vqS, fs, uP, λP, i, k, m, rk, KK, qk, uk, uk1, ∆uk, 
  vqk, λk, λk1, ∆λk, ck, ak, qk, ηk, dk, Wk, uknorm, uknorm2, dknorm, 
  rknorm, ssu, ssk, sk, ndof, zv, PI, P, n, val, εval, cs, ell, 
  nmax = Length[exitpar], 
  dexit = {}, rexit = {}, εexit = {}, ev, evmin, evmax, 
  Kscn, Im, kmax = 5, 
  dcon = 0, ddiv = 0, dref = 0, ccon = 0, cdiv = 0, cref = 0, 
  Wcon = 0, Wdiv = 0, Wref = 0, 
  Ksing = 1.0^14, 
  ns = Length[modpar], 
  ss = Length[histab], 
  uhis = {}, uprt, nupr = 0, 
  soltab = {}, 
  head, ec = 0}, 
  For[m = 1, m < nmax, m++, 
    msp = modpar[[m]]; 
    nsp = Length[msp]; 
    If[nsp == 2, 
      {key, val} = msp; 
      If[key == "REL", ω = val]; 
      If[key == "LM", ν = val]; 
    ];] 
  rS = r[uS, λS]; 
  KS = K[uS] + ν*Im; 
  qS = q[λS]; 
  zv = Table[0, {ndof}]; 
  vqS = LinearSolve[KS, qS]; 
  fS = Sqrt[1 + vqS.vqS]; 
  uk = uP; 
  λk = λP; 
  PI[val_, n_] := PaddedForm[val, n]; 
  P[val_, n_] := PaddedForm[Chop[val, εchp], {n, n - 2}]; 
  For[k = 0, k < kmax, k++, 
    ∆uk = uk - uS; 
    ∆λk = λk - λS; 
    uknorm2 = ∆uk.∆uk; 
    ssu = uknorm2 + 2*∆λk; 
    sk = Sqrt[ssu]; 
    If[cs == "LC", ck = ∆λk - ell; 
      ak = zv; 
      gk = 1]; 
    If[cs == "SC", ck = uknorm2 - ell^2; 
      ak = 2*sk; 
      gk = 0]; 
    If[cs == "MSC", ck = uknorm - ell; 
      ak = ∆uk/uknorm; 
      gk = 0]; 
    If[cs == "AC", ck = (vqS.∆uk + ηk)/fS - ell; 
      ak = vqS/fS; 
      gk = 1/fS]; 
    If[cs == "HC", ck = ssk - ell^2; 
      ak = 2*sk; 
      gk = 2*∆λk]; 
    If[cs == "MHC", ck = sk - ell; 
      ak = sk/uknorm; 
      gk = ηk/uknorm]; 
    If[cs == "EWC", ck = -qS.∆uk + ell; 
      ak = -qS; 
      gk = 0]; 
    Kk = K[uk] + ν*Im; 
    ev = Abs[Eigenvalues[Kk]]; 
    evmin = Min[ev]; 
    evmax = Max[ev]; 
    Kscn = evmax/evmin; 
    If[Kscn > 1.0^14, Break[]]; 
    rk = r[uk, λk]; 
    qk = q[λk]; 
    vqk = LinearSolve[Kk, qk]; 
    ηk = -ck + ak.vqk; 
    λk1 = λk + ηk; 
    dk = vrk + ηk*vqk; 
    uk1 = uk + ηk*dk; 
    dknorm = Sqrt[dk.dk]; 
    Wk = Abs[rk*dk + ηk*uk1.dk]; 
    If[n dig >= 0, uprt = Table[ui[[1]]], {i, nupr}]]; 
    vals = Flatten[{rknorm, dknorm, Wk, ck, λk1, uprt}]; 
    AppendTo[row, P[vals[[1]], ndig]]; 
    AppendTo[soltab, {row}]; 
  ]; 
  For[i = 1, i < Length[vals], i++, 
    AppendTo[row, P[vals[[i]], ndig]]]; 
  AppendTo[soltab, {row}]; 
  ]; 
];

Figure 25.10. The module MatrixNewton that implements a corrector for a system of nonlinear 
  equations with several variables. It is intended to illustrate examples in this Chapter.

§25.8 A MATRIX IMPLEMENTATION
kmax is maximum number of iteration cycles; typically 5 to 10. 
dcon, ddiv, dref used to test against state change: convergence if \( ||d^k||_2 < dcon*dref \),
divergence if \( ||d^k||_2 > ddiv*dref \). If sublist is empty this test is not applied. 
rcon, rdiv, rref used to test against residual: convergence if \( ||r^k||_2 < rcon*rref \),
divergence if \( ||r^k||_2 > rdiv*rref \). If sublist is empty, this test is not applied. 
Wcon, Wdiv, Wref used to test against cycle work: convergence if \( W^k < Wcon*Wref \),
divergence if \( W^k > Wdiv*Wref \), in which \( W^k \) is given by \( () \). If sublist is empty, this
test is not applied. 
Note: if the three foregoing sublists are empty, \( kmax \) cycles will be performed and
ec=0 on exit, unless a stiffness singularity is detected. 
Ksing is a stiffness singularity threshold that causes termination if the spectral con-
dition number of \( K^k \) exceeds \( 1/Ksing \). If omitted the program assumes \( 10^{-14} \). 

\[ \text{modpar} \quad \text{A list of optional modification parameters. Configured as a sequence of sublists:} \]
\[ \text{lb}\{\text{slist1}\},\{\text{slist2}\}, \ldots \}. \] 
Each sublist is headed by a keyword optionally followed by numerical values. Sublists presently implemented:
\[ \{ \text{"REL"}, \omega \} \]: scale each \( d^k \) by relaxation factor \( \omega \in [0, 2] \). Default is \( \omega = 1 \). 
\[ \{ \text{"LM"}, \nu \} \]: add Levenberg-Marquardt correction \( \nu I \) to \( K^k \).
\[ \{ \text{"QB"} \} \]: do quadratic backtracking to find \( \omega^k \) at each cycle (not implemented).

\[ \text{histab} \quad \text{A two-item list:} \{ \text{ndig}, \text{doflist} \} \text{ requesting that a history table be set up to record} \]
\[ \text{and print the iteration progress. Here \( \text{ndig} \) is the number of digits to print after the} \]
decimal point, whereas \( \text{doflist} \) is a list of state DOF freedoms to be saved and
printed. For example, \( \{ 5, \{ 1, 3 \} \} \) asks that components 1 and 3 of the state vector
be printed with 5 decimal digits. If argument is the empty list, the table is not set up. 
The function returns \{ \{ uC, \lambda C \}, ec \}, in which
\[ uC, \lambda C \quad \text{The exit state-control pair.} \]
\[ ec \quad \text{Integer error code. If \( ec=k>0 \), convergence was achieved after \( k \) cycles. Return \( k=0 \)
indicates no convergence or divergence after \( kmax \) cycles. Fatal errors are flagged}
with negative values: \( k=-1 \): iteration diverges; \( k=-2 \): stiffness appears singular. 

\[ \text{§25.9. The Ordinary Newton Method} \]
The Newton iteration discussed in the mathematical literature on solving nonlinear systems assume
that \( \lambda \) is held constant. This corresponds to \( \lambda \) control or load control in our terminology. As
previously noted, fixing \( \lambda \) makes critical points impassable. However, the resultant method provides
good examples to watch the typical behavior of the Newton iteration process. 
If \( \lambda \) is kept constant the incremental step constraint is \( \Delta \lambda_n = \ell_n \), which has derivatives \( 0 \) and 1
with respect to \( u \) and \( \lambda \), respectively. The linear system (25.15) simplifies to
\[ \begin{bmatrix} K & -q \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d \\ \eta \end{bmatrix} = - \begin{bmatrix} r \\ 0 \end{bmatrix} \] 
in which \( c = 0 \) because the constraint is satisfied exactly. Since \( \eta = 0 \), the “bordering” disappears,
reducing (25.40) to \( Kd = -r \). Consequently the Newton iteration becomes
\[ u^{k+1} = u^k - (K^k)^{-1}r^k, \quad \lambda^{k+1} = \lambda^k = \lambda_n + \ell_n \quad \text{(kept fixed)}. \]
Note that the incremental load vector $\mathbf{q}$ disappeared entirely. This is the version found in standard numerical analysis texts, and is that is used in the examples that follow.

**Example 25.8.** The computation of the square root $\sqrt{a}$ of a scalar number $a > 0$ by Newton iteration is set up as follows. The square root satisfies the equation $r(x) = f(x) = x^2 - a = 0$. Starting from an initial value $x_0 > 0$, the Newton iteration computes

$$x^{k+1} = x^k - f(x^k)/f'(x^k) = x^k - ((x^k)^2 - a)/(2x^k).$$

(25.42)

where prime denotes derivative with respect to $x$. The results for $a = 3$ and $x_0 = 1$ are illustrated by the Mathematica program below.

```mathematica
f[x_,a_]:=x^2-a; Df[x_]:=2*x;
a=3.; xk=1.; Print["x0=",xk];
For[k=0, k<6, k++, xkp1=xk-f[xk,a]/Df[xk];
    Print["x",k+1,"=",xkp1//InputForm]; xk=xkp1];
x0=1.
x1=2.
x2=1.75
x3=1.732142857142857143
x4=1.732050810014727541
x5=1.732050807568877295
x6=1.732050807568877294
```

After six cycles the iteration yields 16 places of accuracy for $\sqrt{3}$. Note that the number of exact digits roughly doubles from $k = 2$ onwards. This is typical of the Newton iteration once it “locks in” a root because the process has asymptotically quadratic convergence.

The numerical process illustrated above is actually that used by mathematical software libraries of languages like Fortran or C for the computation of the square root function. However the initial value is determined by a scaled rational interpolant that gives 2-4 digits of accuracy for $x_0$; as a result only 2 or 3 cycles are needed to achieve double-precision accuracy for most inputs.

**Example 25.9.** This example is more typical of a structural application. Consider the residual equation for a 2-DOF system

$$\mathbf{r}(\mathbf{u}, \lambda) = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} u_1 + 2u_1^3 - u_2^2 - 2\lambda \\ 3u_2 - 2u_1u_2 - \lambda \end{bmatrix}$$

(25.43)

The tangent stiffness matrix is

$$\mathbf{K} = \frac{\partial \mathbf{r}}{\partial \mathbf{u}} = \begin{bmatrix} 1 + 6u_1^2 & -2u_2 \\ -2u_2 & 3 - 2u_1 \end{bmatrix}$$

(25.44)

For $\lambda = 1$ the residual equations have three real roots which to six digits of accuracy are

$$\mathbf{u}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \mathbf{u}_2 = \begin{bmatrix} 1.18400 \\ 1.58227 \end{bmatrix}, \quad \mathbf{u}_3 = \begin{bmatrix} 1.66726 \\ -2.98938 \end{bmatrix}.$$  

(25.45)

The following Mathematica programs starts from the initial values $u_1^0 = u_2^0 = 0.8$ and quickly finds the nearest root $u_1 = u_2 = 1$, delivering 16 digits of accuracy after 5 cycles:

```mathematica
ClearAll[r,u1,u2,lambda];
Chapter 25: CONVENTIONAL NEWTON METHODS

\[ \lambda = 1; \]
\[ r[u_1, u_2] := \{u_1 + 2u_1^3 - u_2^2 - 2 - 2\lambda, 3u_2 - 2u_1 - 2u_2 - 2\lambda\}; \]
\[ K_t[u_1, u_2] := \{1 + 6u_1^2, -2u_2, -2u_2, 3 - 2u_1\} \]
\[ u_k := \{0.8, 0.8\}; \]
\[ \text{Print["Starting } v_0 = \", u_k//InputForm];} \]

For \( k = 0, k < 5, k++ \),
\[ \{u_1, u_2\} = \{u_k[[1, 1]], u_k[[2, 1]]\}; \]
\[ u_{k+1} = u_k - \text{Inverse}[K_t[u_1, u_2]].r[u_1, u_2]; \]
\[ \text{Print["Cycle } k = \", k, " u_{k+1} = \", u_{k+1}//InputForm];} \]
\[ u_k = u_{k+1}; \]
\[ \{u_1, u_2\} = \{u_k[[1, 1]], u_k[[2, 1]]\}; \]
\[ \text{Print["Final residual = \", r[u_1, u_2];} \]

Starting \( v_0 = \{0.8, 0.8\} \)
Cycle \( k = 0 \) \( u_1 = \{1.025426944971537002\}, \{0.971916508538894309\}\}
Cycle \( k = 1 \) \( u_2 = \{1.001827210881738689\}, \{1.005246766090385063\}\}
Cycle \( k = 2 \) \( u_3 = \{0.99998431106495672, 0.9999939810418404\}\}
Cycle \( k = 3 \) \( u_4 = \{0.999999995779087286\}, \{0.999999995786530219\}\}
Cycle \( k = 4 \) \( u_5 = \{0.9999999999999999893\}, \{0.9999999999999999659\}\}
-18 -17
Final residual = \{\{-7.48099 10^{-18}\}, \{-1.25767 10^{-17}\}\}

But changing the initial values to \( u_0 = 0.8 \) and \( u_1 = 1.1 \), which is even closer to the \( u_1 = u_2 = 1 \) root, the process converges to the second root in (25.45), reaching 16 digits of accuracy after 8 cycles:

\[ \text{ClearAll}[r, u_1, u_2, \lambda]; \]
\[ \lambda = 1; \]
\[ r[u_1, u_2] := \{u_1 + 2u_1^3 - u_2^2 - 2 - 2\lambda, 3u_2 - 2u_1 - 2u_2 - 2\lambda\}; \]
\[ K_t[u_1, u_2] := \{1 + 6u_1^2, -2u_2, -2u_2, 3 - 2u_1\} \]
\[ u_k := \{0.8, 1.1\}; \]
\[ \text{Print["Starting } u_0 = \", u_k//InputForm];} \]

For \( k = 0, k < 5, k++ \),
\[ \{u_1, u_2\} = \{u_k[[1, 1]], u_k[[2, 1]]\}; \]
\[ u_{k+1} = u_k - \text{Inverse}[K_t[u_1, u_2]].r[u_1, u_2]; \]
\[ \text{Print["Cycle } k = \", k, " u_{k+1} = \", u_{k+1}//InputForm];} \]
\[ u_k = u_{k+1}; \]
\[ \{u_1, u_2\} = \{u_k[[1, 1]], u_k[[2, 1]]\}; \]
\[ \text{Print["Final residual = \", r[u_1, u_2];} \]

Starting \( u_0 = \{0.8, 1.1\} \)
Cycle \( k = 0 \) \( u_1 = \{1.1886363636363636, 1.325\}\}
Cycle \( k = 1 \) \( u_2 = \{1.215044904251438747, 1.718220285975100653\}\}
Cycle \( k = 2 \) \( u_3 = \{1.1897441390194791, 1.602104015495415557\}\}
Cycle \( k = 3 \) \( u_4 = \{1.184187697535010493, 1.582880662857412937\}\}
Cycle \( k = 4 \) \( u_5 = \{1.183998614947572015, 1.582271181104649434\}\}
Cycle \( k = 5 \) \( u_6 = \{1.183998417328768284, 1.582270566284133866\}\}
Cycle \( k = 6 \) \( u_7 = \{1.183998417328558547, 1.582270566283474426\}\}
Cycle \( k = 7 \) \( u_8 = \{1.183998417328558548, 1.582270566283474426\}\}
This illustrates the “finicky” nature of Newton iteration. It can do (and often does) the unexpected, such as diverging or converging to the “wrong” root. In fact the a whole subset of fractal mathematics is devoted to the understanding of “domains of attraction” of roots. Because of this capricious behavior, in practical use of the Newton corrector numerous safeguards are implemented to avoid surprises. But the whole subject is too lengthy for coverage in an introductory treatment.

**Notes and Bibliography**

As indicated in Chapter 18, purely incremental methods dominated the first decade of FEM work for nonlinear applications. Corrective methods appeared in the mid 1960s but were not seriously used in production programs until the 1970s. A brief history of this early period is given by Crisfield in his 1991 textbook [173].

The use of auxiliary systems for Newton iteration under displacement control was introduced by Batoz and Dhatt in 1979 [60]. It was extended to arbitrary increment constraints, as well as critical points, in [223,225]. In these papers a singular \( K \) is stabilized (regularized) by penalty springs. Further extensions of this idea to “floating” structures and substructures may be found in [240,248].
Chapter 25: CONVENTIONAL NEWTON METHODS

Homework Exercises for Chapter 25

Conventional Newton Methods

EXERCISE 25.1 [A:10] If the case of load control, the incremental stepsize constraint reduces to
\[ c = \Delta \lambda_n - \ell_n = 0 \]  
(E25.1)
in which \( \ell_n \) is prescribed. After the predictor step, \( c = 0 \). Show that the Newton method reduces to a standard form that requires only the solution of the first of the auxiliary systems (25.17), and that stiffness augmentation is thus unnecessary.

EXERCISE 25.2 [A:15] The (unscaled) hyperspherical increment control strategy (ICS) of Crisfield [166] relies on the constraint
\[ c = \Delta \lambda_n^2 + \Delta u_n^T \Delta u_n - \ell_n^2 = 0, \]  
(E25.2)
in which \( \ell_n \) is prescribed. Write down the form of the augmented stiffness matrix in (25.15) for the Newton iteration using this constraint.

EXERCISE 25.3 [A:20] The external work control strategy (WC) uses an increment control strategy (ICS) constraint given by
\[ c = \ell_n - q_n^T \Delta u_n = 0, \]  
(E25.3)
in which \( \ell_n \) now has dimensions of energy. (This may be converted back to a length through appropriate scaling by a reference load \( q_{ref} \).) The idea is to specify the linearized external work increment to be constant. Show that this ICS possesses the following advantages and disadvantages:

1. Advantage: It maintains symmetry in the augmented stiffness matrix (25.15); thus a symmetric equation solver may still be used.
2. It completely collapses if \( q_n = 0 \), which occurs if a structure is moving rigidly under a constant load (i.e., \( r(u, \lambda) \) does not depend on \( \lambda \)).

EXERCISE 25.4 [A:35] (Advanced, may take days, worth publishing) Derive the expression of the augmented system (25.15) if the orthogonal trajectory accession method of Fried [293] is used and write out the Newton correction algorithm for this case. Note: the paper [293] is available online; do a Google search for it, download and print.

(Note: don’t do the following Exercises until GeNoBeC.nb is posted in the Index of Chapter 25 — it should be there by the middle of Spring Break, since the corrective modules need to be tested.)

EXERCISE 25.5 [C:15] Download the notebook GeNoBeC.nb linked to the Chapter 23 Index. Initialize the notebook and run the script in the next-to-last cell, which specifies the Mises Truss benchmark (MT), Forward Euler (FE) and Load Control (LC). Setting \( k_{max} = 0 \) suppresses corrector iterations so the program effectively runs as a Purely Incremental (PI) nonlinear solver. After crossing the first limit point, the results should be nonsensical. Then run with \( k_{max} = 3 \), which specify 3 Newton iterations per step. Do results improve? Does changing LC to AC change things significantly?

4 It also collapses in the vicinity of a bifurcation point, but that is more difficult to work out.
EXERCISE 25.6 [C:15] Start as above, but now run the script in the last cell, which specifies the Mises Truss benchmark (MT), Explicit Midpoint Rule (EMR) and Arclength Control (AC). Setting $k_{\text{max}}=0$ suppresses corrector iteration so the program runs as a PI nonlinear solver. Then run with $k_{\text{max}}=3$, which specifies 3 Newton iterations per step. Comment on the results, especially whether the drift error, as displayed by the $||r||$ versus step plot, improves. Notes: (1) the stepsize, as specified by $\varepsilon_{11}$, is larger than in the previous Exercise; (2) the drift error should be zero to 6 or more places once $k_{\text{max}}$ is 2 or higher.

Remark: The Newton arclength-correction logic in $\text{INLSolver}$ contains two “not-quite-correct” expressions left over from older Fortran codes — still they do not seem to hurt much. It was felt that fixing the glitches is not worth the time, since $\text{INLSolver}$ will be eventually superseded by $\text{GeNoBeC}$. 