CHAPTER FOUR

Numerical Methods in Kinematics

The governing equations of kinematics developed in Chapter 3 may be assembled and solved to determine the position, velocity, and acceleration of a system, provided adequate driving conditions are specified. While the kinematic equations that determine the position of the system are highly nonlinear, the velocity and acceleration equations are linear and may be solved using matrix factorization and solution techniques that are well suited for computer implementation. In addition to the inherently nonlinear nature of kinematic equations, the large number of variables involved makes it impractical to even write the explicit governing equations for large-scale systems, much less solve them analytically. Numerical methods that permit both computer assembly and solution of kinematic equations are presented in this chapter.

Four distinct modes of kinematic computation are considered. The first is the determination of an assembled configuration of a system, given only estimates for the position and orientation of each component and the governing constraint equations. While the basic theory of system assembly is similar to position analysis, that is, requiring solution of kinematic constraint and driving conditions, it differs from position analysis that is begun from an assembled configuration in that a good estimate of the assembled configuration may not be available. Furthermore, since kinematic position equations are highly nonlinear, a mechanical system may be specified that cannot be physically assembled or for which constraints are not independent, hence leading to severe mathematical and numerical difficulties.

Once an assembled configuration is obtained with independent kinematic and driving constraints, rapidly convergent and efficient numerical methods may be used to solve the kinematic position equations at each instant on a time grid. This is the second mode of kinematic analysis. Once position is known, the velocity equations may be solved and the results used, in conjunction with position information, to evaluate and solve the acceleration equations. These are the third and fourth modes of kinematic analysis.

Prior to launching into a technical treatment of each of the four modes of kinematic analysis, Sections 4.1 and 4.2 are devoted to the important task of organizing computations and assembling the required equations of kinematics.

This aspect of computer-aided kinematic analysis is the key to harnessing the power of the high-speed computer and relieving the engineer of the burden of equation formulation.

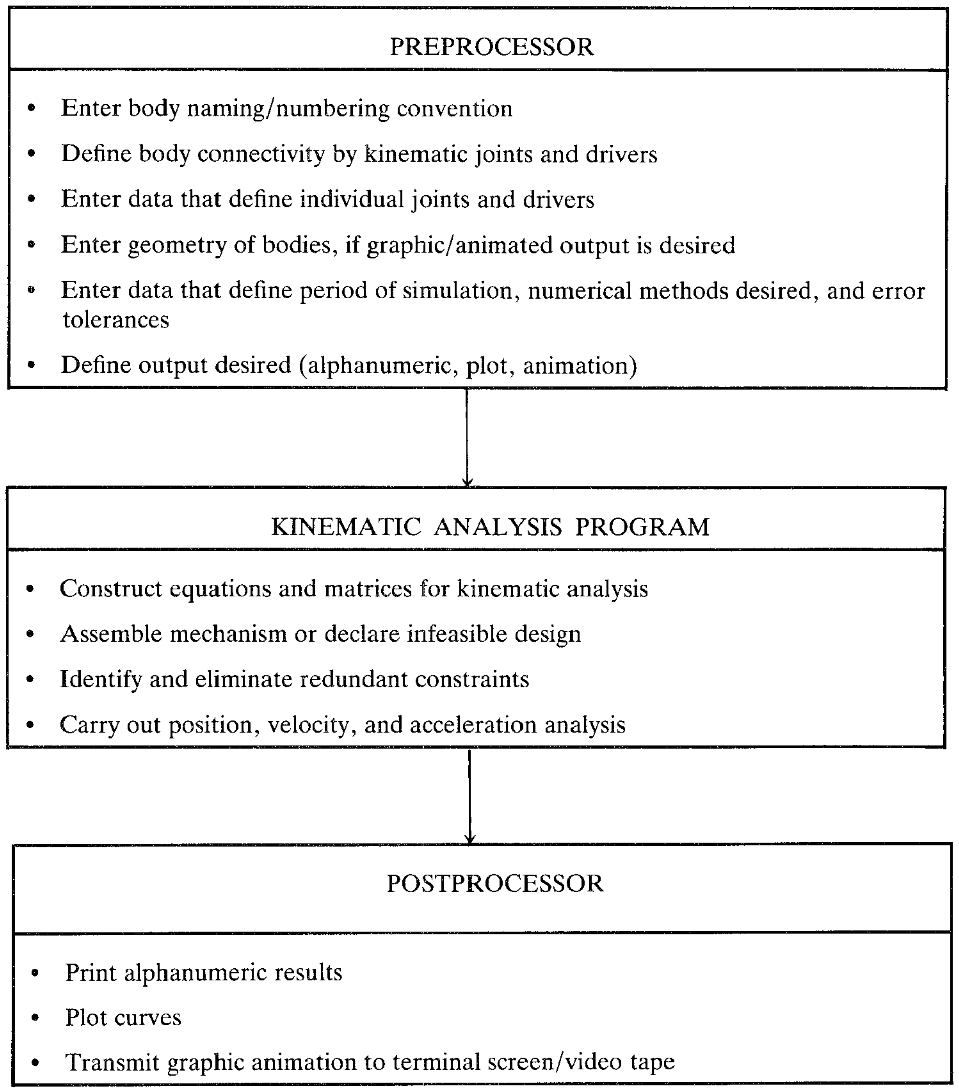
4.1 ORGANIZATION OF COMPUTATIONS

A large-scale kinematics and dynamics computer code, called the Dynamic Analysis and Design System (DADS) [27], has been developed to implement the theory presented in this text for both the kinematic and dynamic analysis of planar and spatial systems. The purpose of this section is to outline the organization of computations within the DADS code that carry out the four phases of kinematic analysis. The DADS code is used to formulate and solve planar kinematics and dynamics examples in Chapters 5 and 8. Spatial kinematics and dynamics applications are studied in Chapters 10 and 12.

Illustrations of the Cartesian coordinate planar kinematics formulation presented in Chapter 3 were limited to small-scale mechanisms. For simple systems, it is practical to manually define body numbering and kinematic constraint equation ordering, evaluate Jacobian matrices and right sides for velocity and acceleration equations, and solve the equations. For large-scale applications, however, even the bookkeeping required for organizing kinematic equations can be quite time consuming and prone to error. The computational flow within the DADS computer code that implements these organizational calculations and forms and solves the resulting equations is briefly summarized in Fig. 4.1.1. For support of kinematic analysis, the DADS code consists of three basic elements: (1) a preprocessor that collects problem definition information and organizes data for computation, (2) a kinematic analysis program that constructs and solves the equations of kinematics, and (3) a postprocessor that prepares output information and displays the results of a kinematic simulation. The DADS code uses many subroutines and modules to support both kinematic and dynamic analysis. Only those functions carried out for kinematic analysis are outlined in this section.

Logic is programmed into the preprocessor to permit the user to enter body names or numbers and define types of joints that connect pairs of bodies. This defines the structure of the kinematic system. The user then enters detailed data for each of the joints, which are used in the formulation of the equations of kinematics. If the user desires a graphical portrayal of motion of the system as output, then the outline geometry of individual bodies must also be entered. Finally, the user enters the desired simulation time interval, defines data that control the numerical solution process, and defines the form of output information desired for subsequent control of the postprocessor. The output of preprocessor computation is a kinematic analysis data set that is read and implemented by the kinematic analysis program.

During kinematic analysis, equations are automatically assembled and



PREPROCESSOR

- Enter body naming/numbering convention
- Define body connectivity by kinematic joints and drivers
- Enter data that define individual joints and drivers
- Enter geometry of bodies, if graphic/animated output is desired
- Enter data that define period of simulation, numerical methods desired, and error tolerances
- Define output desired (alphanumeric, plot, animation)

KINEMATIC ANALYSIS PROGRAM

- Construct equations and matrices for kinematic analysis
- Assemble mechanism or declare infeasible design
- Identify and eliminate redundant constraints
- Carry out position, velocity, and acceleration analysis

POSTPROCESSOR

- Print alphanumeric results
- Plot curves
- Transmit graphic animation to terminal screen/video tape

Figure 4.1.1 DADS computational flow.

solved. The assembly analysis outlined in Section 3.6 is carried out, beginning with estimates provided during preprocessing. If the system assembles, kinematic analysis proceeds. If the system fails to assemble, the user is informed that poor initial estimates have been provided or that an infeasible design has been specified. If the system assembles, computational checks are carried out with the constraint Jacobian to determine if redundant constraints have been defined in the model by evaluating the row rank of the Jacobian $\mathbf{\Phi_q}$. If the Jacobian does not have full row rank, the user is informed that redundant constraints are present and that a reformulation of the model is required. Redundant constraints are automatically removed and, if the proper number of drivers has been specified, analysis is carried out. Once a feasible model is established, position, velocity, and acceleration analysis are executed to predict the kinematic performance of the system over the time interval specified by the user. An output data set that defines the results of analysis is transmitted to the postprocessor for display.

Finally, the postprocessor controls the printing of tabular output and construction of plots of variables of interest, or transmits graphic instructions to a high-speed graphics device that displays an animation of system motion on a terminal screen and/or video tape for subsequent playback and evaluation.

Implementation of the extensive logical and numerical computations indicated by the computational flow in Fig. 4.1.1 requires a large-scale computer code, the details of which are beyond the scope of this text. It is important, however, that the user of such software understand the basic ideas and computational methods that are employed, in order to be able to interpret results and create meaningful and appropriate models that represent reality. Prior to delving into the numerical methods that are used to carry out these computations in later sections of this chapter, it is instructive to review the method that is used to control the flow of computation.

The structure of the DADS kinematic analysis program that carries out the numerical computation is shown in Fig. 4.1.2. The kinematic analysis program consists of three basic elements: (1) an analysis program that controls the process of kinematic analysis, (2) a junction program that assigns tasks to a set of computation modules (subroutines), and (3) a library of modules associated with bodies and joint types that are supported by the formulation.

The analysis program use flags (integers) to define the phase of analysis that is being carried out and passes instructions to the junction subroutine to assemble terms in the required equations during each computational step. The junction subroutine subsequently assigns evaluation tasks to computation modules that evaluate needed terms in the kinematic equations. When these terms are returned to the junction subroutine, it assembles them into arrays that are subsequently passed to the analysis program. The analysis program uses these arrays to carry out computations that solve assigned equations. The process is repeated as kinematic analysis proceeds. The computational aspects of these steps are presented in subsequent sections of this chapter.

A more detailed view of the computational flow and information that is generated during analysis is presented in Fig. 4.1.3. Input data are initially read from the preprocessor and the problem setup phase is initiated under control of the analysis program. Data on individual bodies and joints are passed to modules through the junction subroutine. The modules then accumulate the number of generalized coordinates, the number of constraint equations, and the number and addresses (row and column numbers) of nonzero entries in Jacobian matrices.

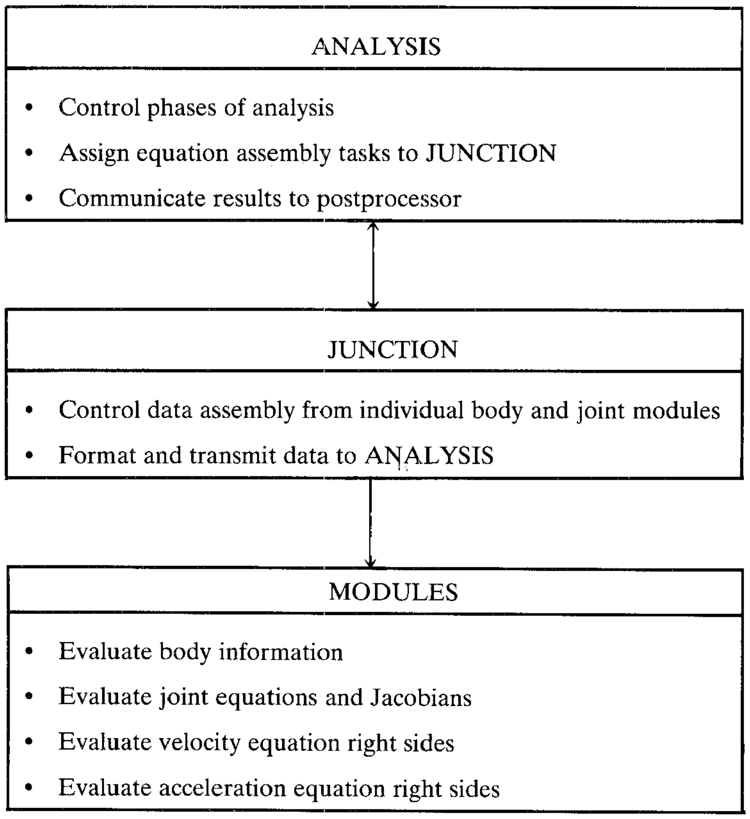

ANALYSIS
• Control phases of analysis
• Assign equation assembly tasks to JUNCTION
• Communicate results to postprocessor
JUNCTION
• Control data assembly from individual body and joint modules
• Format and transmit data to ANALYSIS
MODULES
• Evaluate body information
• Evaluate joint equations and Jacobians
• Evaluate velocity equation right sides
• Evaluate acceleration equation right sides

Figure 4.1.2 Structure of DADS kinematic analysis program.

This information is passed back to analysis through the junction subroutine. The problem setup phase is complete when the analysis program establishes array dimensions and addresses that are required for the execution of kinematic analysis.

The model assembly and feasibility analysis phase is carried out under the control of analysis using the assembly minimization method outlined in Section 3.6. At each iteration of the minimization process, modules generate the constraint equations and Jacobian information that are required for the minimization computation that is presented in Section 4.3. Following successful assembly, an analysis subroutine carries out a computational check on the rank of the constraint Jacobian, using the Gaussian elimination method that is presented in Section 4.4, to identify any redundant constraints that may exist. If a feasible model has been specified, analysis proceeds. If not, the user is informed that refinement of the model or estimated data is required.

Position analysis is carried out at each time step in the analysis program

ANALYSIS	⟷	**JUNCTION/MODULES**
Input Data Reading and Setup		
• Read data • Assign module tasks • Assign array dimensions/ addresses	⟷	• Count generalized coordinates • Count and determine addresses of nonzero Jacobian entries
Model Assembly and Feasibility Analysis		
• Assign module tasks • Assembly minimization computation • Identify redundant constraints	⟷	• Evaluate constraint equations • Evaluate constraint Jacobians
Position Analysis		
• Assign module tasks • Iteratively solve for position	⟷	• Evaluate constraint equations • Evaluate constraint Jacobians
Velocity Analysis		
• Assign module tasks • Solve for acceleration	⟷	• Evaluate constraint Jacobians • Evaluate velocity equation right sides
Acceleration Analysis		
• Assign module tasks • Solve for acceleration	⟷	• Evaluate acceleration equation right sides

Figure 4.1.3 DADS kinematic analysis flow.

using the iterative Newton–Raphson method that is presented in Section 4.5. During each iteration, modules provide information on constraint equation violations and Jacobians. Upon completion of position iteration at a given time step, velocity analysis is initiated, carrying out the computations defined in Section 3.6. During velocity analysis, modules evaluate constraint Jacobian entries and the right side of the velocity equation. Following completion of velocity analysis at the given time step, acceleration analysis is carried out. Modules need provide only the right side of the acceleration equation, since the constraint Jacobian is identical to that constructed during velocity analysis. Upon completion of acceleration analysis, if the final time has been reached, analysis terminates. If not, time is indexed to the next time step and control passes back to position analysis. This process continues until the kinematic simulation is complete.

4.2 EVALUATION OF CONSTRAINT EQUATIONS AND JACOBIAN

When the constraint Jacobian needs to be evaluated, a loop is entered that calls one module for each joint that is defined in the input data, as indicated in Fig. 4.1.2. A vector of nonzero Jacobian entries is evaluated and sent, along with the row and column indexes (pointers), to a subroutine for matrix factoring.

Following assembly, three distinct phases are executed during kinematic analysis over an interval of time. Position analysis involves solving the set of nonlinear constraint equations to find a set of generalized coordinates that satisfies all constraints, as shown in Fig. 4.1.3. A Newton–Raphson algorithm is used to iterate for a solution, taking advantage of the structure of the Jacobian. The following system of linear equations is solved for updated coordinates, the Jacobian is recalculated, and the equations are solved again, until each of the coordinate errors is less than a specified tolerance:

$$\begin{aligned}\mathbf{\Phi}_{\mathbf{q}}^{(i+1)}\Delta\mathbf{q}^{(i)} &= -\mathbf{\Phi}^{(i)}\\ \mathbf{q}^{(i)} &= \mathbf{q}^{(i)} + \Delta\mathbf{q}^{(i)}\end{aligned} \tag{4.2.1}$$

After finding a solution that satisfies constraints within tolerance, the Jacobian must be recalculated prior to solving for velocities from

$$\mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} = -\mathbf{\Phi}_t \equiv \mathbf{v} \tag{4.2.2}$$

The last phase solves the acceleration equations

$$\mathbf{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}} = -[(\mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}})_{\mathbf{q}}\dot{\mathbf{q}} + 2\mathbf{\Phi}_{\mathbf{q}t}\dot{\mathbf{q}} + \mathbf{\Phi}_{tt}] \equiv \boldsymbol{\gamma} \tag{4.2.3}$$

for the acceleration vector. Note that each of these equations uses the same Jacobian matrix. Throughout all calculations, the equation and variable ordering that were established during preprocessing are retained.

A specific example of how the nonzero entry scheme works shows each step involved. For the revolute joint, there are two constraint equations that arise from the vector relation

$$\mathbf{\Phi}^{r(i,j)} \equiv \mathbf{r}_j + \mathbf{A}_j\mathbf{s}_j'^P - (\mathbf{r}_i + \mathbf{A}_i\mathbf{s}_i'^P) = \mathbf{0}$$

or, in scalar form,

$$\begin{aligned}\Phi_1^{r(i,j)} &\equiv x_j + x_j'^P\cos\phi_j - y_j'^P\sin\phi_j - x_i - x_i'^P\cos\phi_i + y_i'^P\sin\phi_i = 0\\ \Phi_2^{r(i,j)} &\equiv y_j + x_j'^P\sin\phi_j + y_j'^P\cos\phi_j - y_i - x_i'^P\sin\phi_i - y_i'^P\cos\phi_i = 0\end{aligned} \tag{4.2.4}$$

The Jacobian for this joint has 8 nonzero entries out of 12 possible Jacobian matrix positions associated with variables of bodies i and j. The nonzero Jacobian entries are

$$\begin{aligned}
&\text{ENTRY}\,(1) = -1.0\\
&\text{ENTRY}\,(2) = x_i'^P \sin\phi_i + y_i'^P \cos\phi_i\\
&\text{ENTRY}\,(3) = 1.0\\
&\text{ENTRY}\,(4) = -x_j'^P \sin\phi_j - y_j'^P \cos\phi_j\\
&\text{ENTRY}\,(5) = -1.0\\
&\text{ENTRY}\,(6) = -x_i'^P \cos\phi_i + y_i'^P \sin\phi_i\\
&\text{ENTRY}\,(7) = 1.0\\
&\text{ENTRY}\,(8) = x_j'^P \cos\phi_j - y_j'^P \sin\phi_j
\end{aligned} \tag{4.2.5}$$

organized as shown in Fig. 4.2.1.

	∂x_i	∂y_i	$\partial \phi_i$		∂x_j	∂y_j	$\partial \phi_j$
$\partial \Phi_1^{r(i,j)}$	(1)		(2)		(3)		(4)
$\partial \Phi_2^{r(i,j)}$		(5)	(6)			(7)	(8)

Figure 4.2.1 Jacobian entries for revolute joint.

Regardless of how many bodies there are in a model, all entries in columns of the Jacobian associated with variables for bodies other than i and j are zero, since these variables do not appear in this constraint equation. If, for example, there are 15 bodies in a system, only 4 of 45 entries in each revolute Jacobian row are nonzero. Thus, only about 10% of the entries are other than zero. Use of this fact saves computer memory and computer time by suppressing operations with zeros.

Example 4.2.1: The four-bar mechanism of Fig. 4.2.2 is made up of four bodies, one of which is ground (body 4). There are four revolute joints, numbered R_1 through R_4, with constraint equations $\mathbf{\Phi}_i^r = 0$, $i = 1, 2, 3$, and 4, of Eq. 4.2.4. Three absolute constraints of Eqs. 3.2.3, 3.2.4, and 3.2.6 are imposed on body 4 to fix it to ground. Finally, an absolute angle driving constraint of Eq. 3.5.3 is imposed on body 1, with $C_3(t) = \omega t$, to cause it to rotate with angular velocity ω (Prob. 4.2.1).

Denoting by $\partial\Phi_1^{ri}$ and $\partial\Phi_2^{ri}$ the rows of the Jacobian corresponding to joint ri, $i = 1, 2, 3$, and 4, placing the Jacobian entries of the ground constraints in rows 9, 10, and 11, and placing the Jacobian entries for the driving constraint

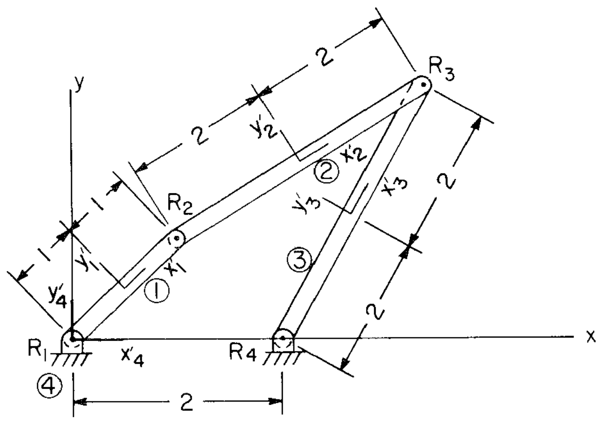


Figure 4.2.2 Four-bar mechanism

in row 12, using Eq. 4.2.5, the Jacobian matrix is (Prob. 4.2.2)

$$
\Phi_{\mathbf{q}} =
\begin{array}{c|cccccccccccc|}
 & \partial x_1 & \partial y_1 & \partial\phi_1 & \partial x_2 & \partial y_2 & \partial\phi_2 & \partial x_3 & \partial y_3 & \partial\phi_3 & \partial x_4 & \partial y_4 & \partial\phi_4 \\
\hline
\partial\Phi_1^{r1} & -1 & 0 & -\sin\phi_1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\partial\Phi_2^{r1} & 0 & -1 & \cos\phi_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\partial\Phi_1^{r2} & -1 & 0 & \sin\phi_1 & 1 & 0 & 2\sin\phi_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
\partial\Phi_2^{r2} & 0 & -1 & -\cos\phi_1 & 0 & 1 & -2\cos\phi_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
\partial\Phi_1^{r3} & 0 & 0 & 0 & -1 & 0 & 2\sin\phi_2 & 1 & 0 & -2\sin\phi_3 & 0 & 0 & 0 \\
\partial\Phi_2^{r3} & 0 & 0 & 0 & 0 & -1 & -2\cos\phi_2 & 0 & 1 & 2\cos\phi_3 & 0 & 0 & 0 \\
\partial\Phi_1^{r4} & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -2\sin\phi_3 & 1 & 0 & -2\sin\phi_4 \\
\partial\Phi_2^{r4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2\cos\phi_3 & 0 & 1 & 2\cos\phi_4 \\
\partial\Phi^{ax} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\partial\Phi^{ay} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\partial\Phi^{a\phi} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\partial\Phi^{a\phi d} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
\end{array}
\qquad \textbf{(4.2.6)}
$$

Note that only 34 of the 144 entries in this 12×12 matrix are nonzero.

The right side $\mathbf{v}$ of the velocity equation of Eq. 4.2.2 has $v_{12} = \omega$ and $v_i = 0$, $i = 1, \ldots, 11$. Since the last four constraints are linear in $\mathbf{q}$ and t, $\gamma_i = 0$, $i = 9$, 10, 11, 12, in Eq. 4.2.3. The first eight components of γ in Eq. 4.2.3 are made up of contributions from the four revolute joint constraints, which may be evaluated using Eq. 4.2.4 (Prob. 4.2.3).

4.3 ASSEMBLY OF A SYSTEM

Recall the system constraint equations of Eq. 3.6.3 at the initial time t_0:

$$\boldsymbol{\Phi}(\mathbf{q}, t_0) = \begin{bmatrix} \boldsymbol{\Phi}^K(\mathbf{q}) \\ \boldsymbol{\Phi}^D(\mathbf{q}, t_0) \end{bmatrix} = \mathbf{0} \qquad \textbf{(4.3.1)}$$

The number of kinematic and driving constraints is normally selected to be equal to the number of generalized coordinates in the system. If the constraints are independent, the Jacobian will be nonsingular. However, it is possible to attempt assembly of the system without specifying an adequate number of driving constraints and perhaps to have specified kinematic or driving constraints that are dependent on other constraints that act on the system, in which case the constraint Jacobian will be rank deficient and Newton–Raphson iteration may fail. Furthermore, the system may have a singular configuration at t_0 or it may be impossible to assemble, in which case Newton–Raphson iteration will fail.

Example 4.3.1: The elementary two-body model of a slider–crank mechanism studied in Example 3.7.3, shown in Fig. 4.3.1, is driven by the condition $\phi_1 = \omega t$. Since absolute constraints require that $x_1 = y_1 = y_2 = \phi_2 = 0$, $x_2 = q$ may be taken as the only generalized coordinate, which must satisfy the distance constraint

$$\Phi(q, t) = (q - \cos \omega t)^2 + \sin^2 \omega t - \ell^2 = 0$$

If $\ell < 1$, then no solution exists beyond the time t^* for which $\sin \omega t^* = \ell$ (i.e., when the coupler is vertical). To observe the behavior of Newton–Raphson iteration in attempting to assemble this simple system, let $\ell = \sqrt{2}/2$ and $\omega = 1$. The critical or lock-up time is then $t^* = \pi/4$. For Newton–Raphson iteration,

$$\Phi_q(q, t) = 2(q - \cos t)$$

Computations are carried out for the following three cases:

Case 1: $t_0 = 0$, $q^{(0)} = 2$. Newton–Raphson iteration yields the results of Table

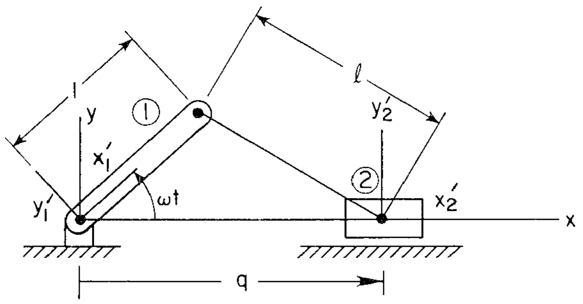


Figure 4.3.1 Two-body slider–crank model.

TABLE 4.3.1 Slider–Crank Results: $t_0 = 0$, $q^{(0)} = 2$

i	$q^{(i)}$	$\Phi(q^{(i)}, 0)$	$\Phi_q(q^{(i)}, 0)$	$\Delta q^{(i)}$	$\left\lvert q^{(i)} - \frac{2+\sqrt{2}}{2} \right\rvert$
0	2.000000	0.500000	2.000000	−0.250000	0.292893
1	1.750000	0.062500	1.500000	−0.041667	0.042893
2	1.708333	0.001736	1.416607	−0.001225	0.001226
3	1.707108	0.000000	1.414216	−0.000000	0.000001

TABLE 4.3.2 Slider–Crank Results: $t_0 = \pi/4$, $q^0 = 1$

i	$q^{(i)}$	$\Phi(q^{(i)}, \pi/4)$	$\Phi_q(q^{(i)}, \pi/4)$	$\Delta q^{(i)}$	$\left\lvert q^{(i)} - \frac{\sqrt{2}}{2} \right\rvert$
0	1.000000	0.085786	0.585786	−0.146447	0.292893
1	0.853553	0.021447	0.292893	−0.073223	0.146447
2	0.780330	0.005362	0.146447	−0.036612	0.073223
3	0.743718	0.001340	0.073223	−0.018306	0.036612
4	0.725413	0.000335	0.036612	−0.009153	0.018306
5	0.716260	0.000084	0.018306	−0.004576	0.009153
6	0.711683	0.000021	0.009153	−0.002288	0.004576
7	0.709395	0.000005	0.004576	−0.001144	0.002288
8	0.708251	0.000001	0.002288	−0.000572	0.001144
9	0.707679	0.000000	0.001144	−0.000286	0.000572
10	0.707393	0.000000	0.000572	−0.000143	0.000286
11	0.707250	0.000000	0.000286	−0.000072	0.000143
12	0.707178	0.000000	0.000143	−0.000036	0.000072
13	0.707143	0.000000	0.000072	−0.000018	0.000036
14	0.707125	0.000000	0.000036	−0.000009	0.000018
15	0.707116	0.000000	0.000018	−0.000004	0.000009
16	0.707111	0.000000	0.000009	−0.000002	0.000004
17	0.707109	0.000000	0.000004	−0.000001	0.000002
18	0.707108	0.000000	0.000003	−0.000001	0.000002
19	0.707108	0.000000	0.000002	−0.000000	0.000001
20	0.707107	0.000000	0.000001	−0.000000	0.000000

4.3.1. Note that the solution error decreases quadratically; that is,

$$\left|q^{(i+1)} - \frac{2+\sqrt{2}}{2}\right| \leq c \left|q^{(i)} - \frac{2+\sqrt{2}}{2}\right|^2$$

where $c \approx 0.06$ and $q = (2+\sqrt{2})/2$ is the solution. This property is typical of Newton–Raphson iteration (see Section 4.5) if $\mathbf{\Phi_q}$ is nonsingular, which is true in this case.

Case 2: $t_0 = \pi/4$, $q^{(0)} = 1$. Newton–Raphson iteration yields the results of Table 4.3.2. Note that even though the Newton–Raphson algorithm converges to the singular solution $q = \sqrt{2}/2$ the rate of convergence is much slower than in case 1. In fact, for singular equations that have a solution, the Newton–Raphson algorithm converges only linearly. For systems of many equations in many unknowns, numerical error may dominate and create very slow convergence or even failure to converge.

Case 3: $t_0 = 3\pi/8$, $q^{(0)} = 1$. Newton-Raphson iteration yields the results of Table 4.3.3. As may have been expected, since there is no solution in this case, the

TABLE 4.3.3 Slider–Crank Results: $t_0 = 3\pi/8$, $q^0 = 1$

i	$q^{(i)}$	$\Phi(q^{(i)}, 3\pi/8)$	$\Phi_q(q^{(i)}, 3\pi/8)$	$\Delta q^{(i)}$
0	1.000000	0.734633	1.234633	−0.595021
1	0.404979	0.354050	0.044590	−7.940071
2	−7.535093	63.044735	−15.835553	3.981215
3	−3.553878	15.850071	−7.873125	2.013187
4	−1.540691	4.052922	−3.846749	1.053597
5	−0.487094	1.110066	−1.739555	0.638132
6	0.151038	0.407213	−0.463291	0.878958
7	1.029996	0.772566	1.294024	−0.596750
8	0.433246	0.356110	0.101125	−3.521482
9	−3.088236	12.400835	−6.941839	1.780391
10	−1.301846	3.191191	−3.369058	0.947206
11	−0.354640	0.897199	−1.474646	0.608416
12	0.253777	0.370170	−0.257814	1.435806
13	1.689582	2.061537	2.613797	−0.788714
14	0.900868	0.622069	1.030370	−0.600238
15	0.300630	0.360286	−0.164107	2.195438

Newton–Raphson algorithm cannot possibly converge. In this case, it simply meanders. In some cases, it may diverge and give computer overflow.

Since the nonlinear constraint equations of Eq. 4.3.1 may contain redundant equations, they may not uniquely determine the position of the system. In fact, they may have no solution. A stable computational method is required to obtain a good estimate of the assembled position of the system or to determine that no solution exists. The basic idea employed is to minimize error in satisfying Eq. 4.3.1, to find a solution of Eq. 4.3.1 that is as near as possible to the initial estimate $\mathbf{q}^0$. The estimate $\mathbf{q}^0$ of the assembled configuration that is provided by the user should be as reasonably accurate. The user may employ experimental data or graphical estimates of the position and orientation of each body in the system to create an estimate $\mathbf{q}^0$ of the assembled configuration.

As outlined in Section 3.6, the objective is to minimize

$$\psi(\mathbf{q}, t_0, r) = (\mathbf{q} - \mathbf{q}^0)^T(\mathbf{q} - \mathbf{q}^0) + r\mathbf{\Phi}^T(\mathbf{q}, t_0)\mathbf{\Phi}(\mathbf{q}, t_0) \tag{4.3.2}$$

to obtain an assembled configuration $\mathbf{q}^a$. The parameter $r > 0$ is a weighting constant that places relative emphasis on deviation from the initial estimate $\mathbf{q}^0$ and solution of Eq. 4.3.1. Since a solution of Eq. 4.3.1 is desired, if one exists, the parameter r in Eq. 4.3.2 is allowed to grow to a very large number in order to precisely satisfy the constraint equations. The method employed is to find $\mathbf{q}(r)$ to minimize the function ψ in Eq. 4.3.2 for a given r, using an optimization algorithm. The parameter r is then increased and the minimization problem is solved again using $\mathbf{q}(r)$ from the previous solution as a starting point. Incrementally increasing r is important, since if the optimization computation is initiated with a very large value of r, convergence might be difficult to achieve, particularly in cases for which no solution of Eq. 4.3.1 exists. At particular values of r, the function ψ of Eq. 4.3.2 is minimized to obtain $\mathbf{q}(r)$, and the limit as r approaches infinity is approximated to obtain

$$\mathbf{q}^a = \lim_{r \to \infty} \mathbf{q}(r) \tag{4.3.3}$$

In actual computation, r is increased in discrete steps, $r_i > r_{i-1}$, and convergence of the limit of Eq. 4.3.3 is tested by evaluating the difference between $\mathbf{q}(r_i)$ and $\mathbf{q}(r_{i-1})$.

If the sequence $\mathbf{q}(r_i)$ converges, then a check is required to confirm that $\mathbf{q}^a$ satisfies Eq. 4.3.1. If it does, an assembled configuration has been achieved. If it does not, it may be concluded either that the system cannot be assembled or that the iterative optimization algorithm is incapable of converging from the initial estimate of the assembled configuration that was given. In the latter case, the user is encouraged to initiate the minimization process from other estimates. While this is not a precise mathematical process, it can be greatly assisted by engineering intuition and graphically obtained estimates.

If there are fewer than nc (the dimension of $\mathbf{q}$) constraint equations in Eq. 4.3.1, or if there are nc equations and some are dependent, then there may be many solutions of Eq. 4.3.1. In either of these cases, the minimization solution $\mathbf{q}^a$, if it satisfies Eq. 4.3.1, is the solution that minimizes the first term in Eq. 4.3.2. In the least-square sense, it is the solution of Eq. 4.3.1 that is nearest the initial estimate $\mathbf{q}^0$.

A method that is well suited for minimizing the function of Eq. 4.3.2 for large-scale systems is called the *conjugate gradient minimization algorithm* of nonlinear programming [28]. The implementation employed here is presented in reference 28. This and virtually all other iterative optimization methods require the gradient of the function that is to be minimized. For the function ψ in Eq. 4.3.2, this gradient is

$$\psi_{\mathbf{q}} = 2(\mathbf{q} - \mathbf{q}^0)^T + 2r\boldsymbol{\Phi}^T(\mathbf{q}, t_0)\boldsymbol{\Phi}_{\mathbf{q}}(\mathbf{q}, t_0) \tag{4.3.4}$$

It is important to note that evaluation of this gradient requires only the constraint Jacobian and the constraint function. Thus, the computational machinery that is needed to implement the remaining modes of kinematic analysis is directly applicable to carry out numerical minimization in the assembly mode.

The iterative conjugate gradient algorithm employed here was developed by Fletcher and Powell [29] and involves the following sequence of computations [28, 29]:

1. Begin with the estimate $\mathbf{q}^{(1)} = \mathbf{q}^0$ of $\mathbf{q}$ and $\mathbf{H}^{(1)} = \mathbf{H}^0 = \mathbf{I}$ as an estimate of the matrix of second derivatives of ψ.
2. At iteration i, compute

$$\mathbf{s}^i = -\mathbf{H}^{(i+1)}\psi_{\mathbf{q}}^T(\mathbf{q}^{(i)})$$

3. Use a one-dimensional search algorithm [28] to find $\alpha = \alpha^i$ that minimizes $\psi(\mathbf{q}^{(i)} + \alpha\mathbf{s}^i)$.
4. Compute

$$\mathbf{q}^{(i+1)} = \mathbf{q}^{(i)} + \alpha^i\mathbf{s}^i$$
$$\mathbf{H}^{(i+1)} = \mathbf{H}^{(i)} + \mathbf{A}^i + \mathbf{C}^i$$

where

$$\mathbf{A}^i = \left(\frac{\alpha^i}{\mathbf{s}^{iT}\mathbf{y}^i}\right)\mathbf{s}^i\mathbf{s}^{iT}$$
$$\mathbf{y}^i = \psi_{\mathbf{q}}^T(\mathbf{q}^{(i+1)}) - \psi_{\mathbf{q}}^T(\mathbf{q}^{(i)})$$
$$\mathbf{C}^i = -\left(\frac{1}{\mathbf{y}^{iT}\mathbf{H}^{(i)}\mathbf{y}^i}\right)\mathbf{H}^{(i)}\mathbf{y}^i\mathbf{y}^{iT}\mathbf{H}^{(i)}$$

5. If $\psi_{\mathbf{q}}(\mathbf{q}^{(i+1)}) = \mathbf{0}$ or if $\mathbf{q}^{(i+1)} - \mathbf{q}^{(i)}$ is sufficiently small, terminate. Otherwise, return to step 2 with $i \to i + 1$.

Examples in which this algorithm is used for the assembly of large-scale systems are given in Chapters 5 and 10.

4.4 LINEAR EQUATION SOLUTION AND MATRIX FACTORIZATION

Matrix equations arise in both the theory and numerical solution of kinematics problems. The rank of the constraint Jacobian provides information on the number of degrees of freedom of the system and must usually be determined numerically. Velocity and acceleration equations derived in Chapter 3 are matrix equations that must be solved numerically. Finally, the position equations are generally nonlinear, but are solved iteratively through solution of a sequence of linear equations (see Section 4.5).

The purpose of this section is to provide an introduction to the analysis and solution of matrix equations. The reader who is interested in more detail is referred to the literature on matrix theory and numerical methods (e.g., [22], [30]).

4.4.1 Gaussian Methods

Consider a system of n linear algebraic equations in n unknowns, with real constant coefficients,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{4.4.1}$$

which can be written in matrix form as

$$\mathbf{Ax} = \mathbf{b} \tag{4.4.2}$$

where

$$\begin{aligned} \mathbf{x} &= [x_1, x_2, \ldots, x_n]^T \\ \mathbf{b} &= [b_1, b_2, \ldots, b_n]^T \\ \mathbf{A} &= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{2n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & & \vdots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \end{aligned}$$

There are many methods of solving Eq. 4.4.1. Cramer's rule [22] is one of the best known but most inefficient methods. Among the more efficient methods is *Gaussian elimination*, which is based on the elementary idea of eliminating variables one at a time. The Gaussian elimination method consists of two major steps, *forward elimination* and *back substitution.*

Forward Elimination First, make the coefficient of x_1 in the first equation unity by dividing that equation by a_{11} (presume for now that $a_{11} \neq 0$). Then eliminate the variable x_1 from the jth equation by multiplying the first modified equation by $-a_{j1}$ and adding the resulting equation to the jth equation, $j = 2, \ldots, n$, to obtain

$$\begin{bmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & & & \vdots & \\ 0 & a_{n2}^{(1)} & a_{n3}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix} \tag{4.4.3}$$

Next, make the coefficient of x_2 in the second equation of Eq. 4.4.3 equal to unity by dividing that equation by $a_{22}^{(1)}$, presuming for now that it is not zero. Then eliminate x_2 from the jth equation by adding $-a_{j2}$ times the second modified equation to the jth equation, $j = 3, \ldots, n$, to obtain

$$\begin{bmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & 1 & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{(33)}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & & & & \\ 0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ \\ b_n^{(2)} \end{bmatrix} \tag{4.4.4}$$

After $n-1$ steps of this algorithm, divide the nth row by $a_{nn}^{(n-1)}$, presuming for now that it is not zero, to obtain the final result of the forward-elimination step:

$$\begin{bmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & 1 & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & 1 & \cdots & a_{3n}^{(3)} \\ \vdots & & & & \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ \\ b_n^{(n)} \end{bmatrix} \tag{4.4.5}$$

Observe that all elements below the diagonal of the matrix are zero, so the determinant of the matrix is 1; hence it is nonsingular.

Recall that to complete calculations leading to Eq. 4.4.5, it was presumed that the modified diagonal elements $a_{jj}^{(j-1)}$, obtained after $j-1$ steps, are not zero. Refinements in this algorithm are presented later in this section to overcome this limitation.

Back Substitution Back substitution consists of $n-1$ elementary solution steps, using Eq. 4.4.5. From the nth equation of Eq. 4.4.5, $x_n = b_n^{(n)}$. This value of x_n may be substituted into equation $n-1$. Equation $n-1$ then gives the value of x_{n-1}. This process is repeated, using the preceding equations, to solve for the remaining variables.

Example 4.4.1: Solve the matrix equation

$$\begin{bmatrix} 3 & 1 & -1 \\ -1 & 2 & 1 \\ 2 & -3 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ -1 \end{bmatrix}$$

Forward elimination yields

1.

$$\begin{bmatrix} 1 & \frac{1}{3} & -\frac{1}{3} \\ -1 & 2 & 1 \\ 2 & -3 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ 6 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{7}{3} & \frac{2}{3} \\ 0 & -\frac{11}{3} & \frac{5}{3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{20}{3} \\ -\frac{7}{3} \end{bmatrix}$$

2.

$$\begin{bmatrix} 1 & \frac{1}{3} & -\frac{1}{3} \\ 0 & 1 & \frac{2}{7} \\ 0 & -\frac{11}{3} & \frac{5}{3} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{20}{7} \\ -\frac{7}{3} \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{1}{3} & -\frac{1}{3} \\ 0 & 1 & \frac{2}{7} \\ 0 & 0 & \frac{19}{7} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{20}{7} \\ \frac{57}{7} \end{bmatrix}$$

3.

$$\begin{bmatrix} 1 & \frac{1}{3} & -\frac{1}{3} \\ 0 & 1 & \frac{2}{7} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ \frac{20}{7} \\ 3 \end{bmatrix}$$

Back substitution yields

1. $x_3 = 3$
2. $x_2 + \frac{2}{7}(3) = \frac{20}{7} \rightarrow x_2 = 2$
3. $x_1 + \frac{1}{3}(2) - \frac{1}{3}(3) = \frac{2}{3} \rightarrow x_1 = 1$

In forward elimination, it is clear that the algorithm fails at the jth step if the *pivot element* $a_{jj}^{(j-1)}$ is zero. Also, when the pivot element $a_{jj}^{(j-1)}$ becomes very small, round-off error in computation may lead to erroneous results. Therefore, the order in which the equations are treated during forward elimination may significantly affect the accuracy of the solution obtained. To circumvent this difficulty, the ordering of equations is determined by the algorithm.

Row Pivoting In the jth forward-elimination step of Gaussian elimination, the equation with the largest coefficient (in absolute value) of x_j on or below the diagonal of the coefficient matrix is chosen. Before the elimination step, this row of the matrix and the right-side vector are interchanged with the jth row of the matrix and right side. This procedure is called *row pivoting* and amounts simply to interchanging the order of equations, which does not change the solution. The following example illustrates this procedure.

Example 4.4.2: Perform Gaussian elimination with row pivoting to solve the following set of equations:

$$\begin{bmatrix} 4 & -3 & 5 & 2 \\ -3 & 1 & 1 & -6 \\ 5 & -5 & 10 & 0 \\ 2 & -3 & 9 & -7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 9 \\ 2.5 \\ 13.5 \end{bmatrix}$$

(1) The largest coefficient in column 1 is 5. Interchanging the first and third equations,

$$\begin{bmatrix} 5 & -5 & 10 & 0 \\ -3 & 1 & 1 & -6 \\ 4 & -3 & 5 & 2 \\ 2 & -3 & 9 & -7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2.5 \\ 9 \\ -1.5 \\ 13.5 \end{bmatrix}$$

Forward elimination now yields

$$\begin{bmatrix} 1 & -1 & 2 & 0 \\ 0 & -2 & 7 & -6 \\ 0 & 1 & -3 & 2 \\ 0 & -1 & 5 & -7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 10.5 \\ -3.5 \\ 12.5 \end{bmatrix}$$

(2) The largest coefficient in column 2 on or below the diagonal is -2, which is on the diagonal, so no interchange is necessary. Forward elimination yields

$$\begin{bmatrix} 1 & -1 & 2 & 0 \\ 0 & 1 & -3.5 & 3 \\ 0 & 0 & 0.5 & -1 \\ 0 & 0 & 1.5 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -5.25 \\ 7.25 \\ 1.75 \end{bmatrix}$$

(3) The largest coefficient in column 3 on or below the diagonal is in the fourth row. Interchanging the third and fourth equations,

$$\begin{bmatrix} 1 & -1 & 2 & 0 \\ 0 & 1 & -3.5 & 3 \\ 0 & 0 & 1.5 & -4 \\ 0 & 0 & 0.5 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -5.25 \\ 7.25 \\ 1.75 \end{bmatrix}$$

Forward elimination yields

$$\begin{bmatrix} 1 & -1 & 2 & 0 \\ 0 & 1 & -3.5 & 3 \\ 0 & 0 & 1 & -2.66 \\ 0 & 0 & 0 & 0.33 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -5.25 \\ 4.83 \\ -0.66 \end{bmatrix}$$

(4) Finally,

$$\begin{bmatrix} 1 & -1 & 2 & 0 \\ 0 & 1 & 3.5 & 3 \\ 0 & 0 & 1 & -2.66 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -5.25 \\ 4.83 \\ -2 \end{bmatrix}$$

Back substitution yields $\mathbf{x} = [0.5, -1, -0.5, -2]^T$.

Full Pivoting *Full pivoting* is the selection of the largest element (in absolute value) from among the diagonal element and all elements below and to the right of the diagonal element as the pivot for the next stage of forward elimination. In full pivoting, both row and column interchanges are required to bring the largest element to the diagonal. When columns of the coefficient matrix are interchanged, the corresponding variables in vector **x** are interchanged without changing the solution.

Example 4.4.3: Apply Gaussian elimination with full pivoting to solve the following set of equations:

$$\begin{bmatrix} 2 & -1 & 1 \\ -1 & 0 & 2 \\ 1 & 4 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ -5 \end{bmatrix}$$

(1) The largest element in the matrix is 4. Interchanging columns 2 and 1 and the associated variables to bring the largest element into the first column,

$$\begin{bmatrix} -1 & 2 & 1 \\ 0 & -1 & 2 \\ 4 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ -5 \end{bmatrix}$$

Interchanging equations 3 and 1 to bring the largest element to the pivot position,

$$\begin{bmatrix} 4 & 1 & -2 \\ 0 & -1 & 2 \\ -1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} -5 \\ 5 \\ 0 \end{bmatrix}$$

Forward elimination yields

$$\begin{bmatrix} 1 & 0.25 & -0.5 \\ 0 & -1 & 2 \\ 0 & 2.25 & 0.5 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.25 \\ 5 \\ -1.25 \end{bmatrix}$$

(2) The largest eligible coefficient for the second pivot element is 2.25. Since it is in the second column, interchanging equations 2 and 3 yields

$$\begin{bmatrix} 1 & 0.25 & -0.5 \\ 0 & 2.25 & 0.5 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.25 \\ -1.25 \\ 5 \end{bmatrix}$$

Forward elimination yields

$$\begin{bmatrix} 1 & 0.25 & -0.5 \\ 0 & 1 & 0.22 \\ 0 & 0 & 2.22 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.25 \\ -0.55 \\ 4.44 \end{bmatrix}$$

(3) Finally,

$$\begin{bmatrix} 1 & 0.25 & -0.5 \\ 0 & 1 & 0.22 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1.25 \\ -0.55 \\ 2 \end{bmatrix}$$

Back substitution yields $\mathbf{x} = [-1, 0, 2]^T$.

Forward Elimination with Nonsquare Matrices The forward-elimination step of the Gaussian method, with or without pivoting, can be applied to nonsquare matrices. Consider the case of m equations in n unknowns, with $m < n$; that is,

$$\begin{aligned} a_{11}x_1 + \cdots + a_{mn}x_n &= b_1 \\ &\vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n &= b_m \end{aligned} \tag{4.4.6}$$

If no zero pivots are encountered in forward elimination, after m steps the system may be reduced to the form

$$\begin{bmatrix} 1 & a_{12}^{(m)} & \cdots & a_{1m}^{(m)} & a_{1m+1}^{(m)} & \cdots & a_{1n}^{(m)} \\ 0 & 1 & \cdots & a_{2m}^{(m)} & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & & \\ 0 & 0 & \cdots & 1 & a_{mm+1}^{(m)} & \cdots & a_{mn}^{(m)} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_m \\ v_1 \\ \vdots \\ v_{n-m} \end{bmatrix} = \begin{bmatrix} b_1^{(m)} \\ \vdots \\ \\ b_m^{(m)} \end{bmatrix} \tag{4.4.7}$$

where $\mathbf{u} = [u_1, \ldots, u_m]^T$ and $\mathbf{v} = [v_1, \ldots, v_{n-m}]^T$ contain elements of $\mathbf{x}$ that are reordered due to column pivoting. Equation 4.4.7 may be written in partitioned form as

$$\mathbf{U}\mathbf{u} + \mathbf{R}\mathbf{v} = \hat{\mathbf{b}} \tag{4.4.8}$$

where

$$\mathbf{U} = \begin{bmatrix} 1 & a_{12}^{(m)} & \cdots & a_{1m}^{(m)} \\ 0 & 1 & \cdots & a_{2m}^{(m)} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} a_{1m+1}^{(m)} & \cdots & a_{1n}^{(m)} \\ \vdots & & \vdots \\ a_{mm+1}^{(m)} & \cdots & a_{mn}^{(m)} \end{bmatrix} \tag{4.4.9}$$

$$\hat{\mathbf{b}} = \begin{bmatrix} b_1^{(m)} \\ \vdots \\ b_m^{(m)} \end{bmatrix}$$

Note that for any value of $\mathbf{v}$ in Eq. 4.4.8 $\mathbf{u}$ may be determined through back substitution, that is, from

$$\mathbf{U}\mathbf{u} = \hat{\mathbf{b}} - \mathbf{R}\mathbf{v}$$

Thus, $\mathbf{v}$ may be treated as a vector of *independent variables* and $\mathbf{u}$ as a vector of *dependent variables*. The variable $\mathbf{x}$ is said to be *partitioned* into independent and dependent coordinates.

The factorization of Eq. 4.4.7 is possible, using full pivoting, if and only if the coefficient matrix $\mathbf{A}$ of Eq. 4.4.6 has full row rank [22]. Thus, a foolproof method of determining the rank of $\mathbf{A}$ is to carry out Gaussian forward elimination, using full pivoting, to obtain

$$\begin{bmatrix} 1 & a_{12}^{(r)} & \cdots & a_{1r}^{(r)} & a_{1r+1}^{(r)} & \cdots & a_{1n}^{(r)} \\ 0 & 1 & \cdots & a_{2r}^{(r)} & a_{2r+1}^{(r)} & \cdots & a_{2n}^{(r)} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & a_{rr+1}^{(r)} & \cdots & a_{rn}^{(r)} \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_r \\ v_1 \\ \vdots \\ v_{n-r} \end{bmatrix} = \begin{bmatrix} b_1^{(r)} \\ \vdots \\ b_r^{(r)} \\ b_{r+1}^{(r)} \\ \vdots \\ b_m^{(r)} \end{bmatrix} \tag{4.4.10}$$

In partitioned matrix form, this is

$$\begin{aligned} \mathbf{U}_r\mathbf{u} + \mathbf{R}_r\mathbf{v} &= \hat{\mathbf{b}}_r \\ \mathbf{0} &= \hat{\mathbf{b}}_{m-r} \end{aligned} \tag{4.4.11}$$

where $\mathbf{U}_r$, $\mathbf{R}_r$, and $\hat{\mathbf{b}}_r$ are defined as in Eq. 4.4.9, with m replaced by r, and

$$\hat{\mathbf{b}}_{m-r} = \begin{bmatrix} b_{r+1}^{(r)} \\ \vdots \\ b_m^{(r)} \end{bmatrix} \tag{4.4.12}$$

Thus, the row rank of matrix $\mathbf{A}$ is r and, if $r < m$, the second of Eqs. 4.4.11 shows that Eq. 4.4.10, and hence Eq. 4.4.6, has a solution if and only if $\hat{\mathbf{b}}_{m-r} = \mathbf{0}$. If $\hat{\mathbf{b}}_{m-r} = \mathbf{0}$, then the equations of Eq. 4.4.6 that were interchanged into the last $m - r$ equations (rows) of Eq. 4.4.7 are dependent and may be eliminated.

Example 4.4.4: Use Gaussian forward elimination to determine the rank and a dependent and independent variable partitioning for the following set of equations:

$$\mathbf{Ax} \equiv \begin{bmatrix} 1 & 0 & 2 & 1 & 4 \\ 1 & 1 & 2 & 0 & 3 \\ 3 & 1 & 6 & 2 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 8 \end{bmatrix} \equiv \mathbf{b}$$

The solution will be carried out without pivoting until a zero diagonal entry appears that requires pivoting.

1. Forward elimination in the first column yields

$$\begin{bmatrix} 1 & 0 & 2 & 1 & 4 \\ 0 & 1 & 0 & -1 & -1 \\ 0 & 1 & 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \\ -1 \end{bmatrix}$$

2. Forward elimination in the second column yields

$$\begin{bmatrix} 1 & 0 & 2 & 1 & 4 \\ 0 & 1 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ -1 \\ 0 \end{bmatrix}$$

Thus, the rank of the coefficient matrix is 2 and, since $b_3 = 0$, the equations are consistent. The partitioning defined by this process is $\mathbf{u} = [x_1, x_2]^T$ and $\mathbf{v} = [x_3, x_4, x_5]^T$. The solution for x_1 and x_2, for any values of x_3, x_4, and x_5, is thus

$$x_1 = 3 - 2x_3 - x_4 - 4x_5$$
$$x_2 = -1 + x_4 + x_5$$

If a different right-side vector had been chosen, the rank would still be 2, but the equations could be *inconsistent*; that it, there may exist no solution (Prob. 4.4.4). If full pivoting had been employed in the forward-elimination process, the rank of $\mathbf{A}$ would still be determined to be 2, but a different partitioning of variables would result (Prob. 4.4.5).

On a digital computer, the bottom right block of zeros in the coefficient matrix of Eq. 4.4.10 will not contain precisely zero entries. The entries will be zero only to within the round-off error that has accumulated during forward elimination. Similarly, $\hat{\mathbf{b}}_{m-r}$ can only be zero to within the same numerical precision.

A perplexing and numerically challenging situation arises when the bottom right factors in the coefficient matrix in Eq. 4.4.10 are nearly zero, but not to within round-off error. If the entries in $\hat{\mathbf{b}}_{m-r}$ are a few orders of magnitude larger than elements in the corresponding rows of the matrix, numerical difficulties arise. In this case, the coefficient matrix is said to be *ill-conditioned,* and severe numerical difficulties may be encountered.

4.4.2 L–U Factorization

While the Gaussian methods of Section 4.4.1 are easy to understand and are reliable, better methods exist as regards efficiency and computer memory utilization. One such method is introduced here. Given any nonsingular matrix $\mathbf{A}$, there exists [22] an upper triangular matrix $\mathbf{U}$ with nonzero diagonal elements and a lower triangular matrix $\mathbf{L}$ with unit diagonal elements, such that

$$\mathbf{A} = \mathbf{L}\mathbf{U} \tag{4.4.13}$$

Factorization of $\mathbf{A}$ into the product $\mathbf{LU}$ is called *L–U factorization.* Once the $\mathbf{L}$ and $\mathbf{U}$ factors are obtained, by whatever method, the equation

$$\mathbf{A}\mathbf{x} = \mathbf{L}\mathbf{U}\mathbf{x} = \mathbf{b} \tag{4.4.14}$$

is solved by transforming it into

$$\mathbf{L}\mathbf{y} = \mathbf{b} \tag{4.4.15}$$

and

$$\mathbf{U}\mathbf{x} = \mathbf{y} \tag{4.4.16}$$

Equation 4.4.15 is solved first for $\mathbf{y}$ and then Eq. 4.4.16 is solved for $\mathbf{x}$. Since both Eqs. 4.4.15 and 4.4.16 have triangular coefficient matrices, their solutions are easily obtained by back substitution.

Crout's method calculates elements of $\mathbf{L}$ and $\mathbf{U}$ recursively. To illustrate how Crout's method generates elements of $\mathbf{L}$ and $\mathbf{U}$, consider a matrix $\mathbf{A}$ of rank $n = 3$ that requires no row or column interchanges (i.e., no pivoting). Matrix $\mathbf{A}$

can be written as

$$\begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{4.4.17}$$

An *auxiliary matrix* **B** can be defined, consisting of elements of **L** and **U**, such that

$$\mathbf{B} \equiv \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21} & u_{22} & u_{23} \\ \ell_{31} & \ell_{32} & u_{33} \end{bmatrix} \tag{4.4.18}$$

Elements of **B** are to be calculated in the order indicated by the diagram

$$\begin{Bmatrix} ① & ② & ③ \\ ④ & ⑥ & ⑦ \\ ⑤ & ⑧ & ⑨ \end{Bmatrix} \tag{4.4.19}$$

where Ⓚ indicates the kth element to be calculated. The elements of **L** and **U** are calculated simply by equating elements in Eq. 4.4.17 to a_{jk} successively, according to the order shown in Eq. 4.4.19, using the product of the jth row of **L** and the kth column of **U**. For a 3×3 matrix, Eq. 4.4.17 yields

① $a_{11} = u_{11}$, ② $a_{12} = u_{12}$, ③ $a_{13} = u_{13}$

④ $a_{21} = \ell_{21}u_{11}$, so $\ell_{21} = a_{21}/u_{11}$

⑤ $a_{31} = \ell_{31}u_{11}$, so $\ell_{31} = a_{31}/u_{11}$

⑥ $a_{22} = \ell_{21}u_{12} + u_{22}$, so $u_{22} = a_{22} - \ell_{21}u_{12}$

⑦ $a_{23} = \ell_{23} = \ell_{21}u_{13} + u_{23}$, so $u_{23} = a_{23} - \ell_{21}u_{13}$

⑧ $a_{32} = \ell_{31}u_{12} + \ell_{32}u_{22}$, so $\ell_{32} = (a_{32} - \ell_{31}u_{12})/u_{22}$

⑨ $a_{33} = \ell_{31}u_{13} + \ell_{32}u_{23} + u_{33}$, so $u_{33} = a_{33} - \ell_{31}u_{13} - \ell_{32}u_{23}$

Note that calculation of element Ⓚ of the auxiliary matrix **B**, which is an element of either **L** or **U**, involves only the element of **A** in the same position and elements of **B** that have already been calculated. As element Ⓚ is obtained, it is recorded in matrix **B**. In fact, it may be recorded in the corresponding position of the original **A** matrix, if there is no need to keep **A**.

Crout's method for a general $n \times n$ matrix **A** can be stated as follows [30, 31]:

1. Initially set the iteration counter to $i = 1$.
2. Use the matrix conversion diagram

$$\begin{bmatrix} a_{ii} & \mathbf{a}_{ik}^T \\ \mathbf{a}_{ki} & \mathbf{A}_{kk} \end{bmatrix} \rightarrow \begin{bmatrix} u_{ii} & \mathbf{u}_{ik}^T \\ \mathbf{l}_{ki} & \mathbf{D}_{kk} \end{bmatrix} \tag{4.4.20}$$

to obtain

$$u_{ii} = a_{ii}$$
$$\mathbf{u}_{ik}^T = \mathbf{a}_{ik}^T$$
$$\mathbf{l}_{ki} = \frac{\mathbf{a}_{ki}}{u_{ii}}$$
$$\mathbf{D}_{kk} = \mathbf{A}_{kk} - \mathbf{l}_{ki}\mathbf{u}_{ik}^T$$

where a_{ii} is the ith diagonal element of $\mathbf{A}$, $\mathbf{a}_{ik}^T$ is the part of the ith row of $\mathbf{A}$ to the right of a_{ii}, $\mathbf{a}_{ki}$ is the part of the ith column of $\mathbf{A}$ below a_{ii}, and $\mathbf{A}_{kk}$ is the $(n-i)\times(n-i)$ submatrix to the bottom right of $\mathbf{A}$.

3. Increment i; that is, $i \rightarrow i+1$. If $i=n$, L–U factorization is complete. Otherwise, go to step 2.

Example 4.4.5: Apply L–U factorization to matrix $\mathbf{A}$ and solve the set of algebraic equations $\mathbf{Ax}=\mathbf{b}$ for the unknown $\mathbf{x}$, where

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & -2 \\ -1 & 2.5 & -5 \\ 3 & -4.5 & 0 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ -9.5 \\ 10.5 \end{bmatrix}$$

Following the algorithm, $i=1$:

$$a_{11}=2, \qquad \mathbf{a}_{ik}^T=[1, -2], \qquad \mathbf{a}_{ki}=[-1, 3]^T, \qquad \mathbf{A}_{kk} = -\begin{bmatrix} 2.5 & -5 \\ 4.5 & 0 \end{bmatrix}$$

This step can be performed by overwriting on matrix $\mathbf{A}$ to obtain

$$\begin{bmatrix} 2 & 1 & -2 \\ -1 & 2.5 & -5 \\ 3 & -4.5 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 1 & -2 \\ -0.5 & 3 & -6 \\ 1.5 & -6 & 3 \end{bmatrix}$$

$i=2$: At this step, row 1 and column 1 remain intact. Step 2 applies only to the 2×2 matrix $\begin{bmatrix} 3 & -6 \\ -6 & 3 \end{bmatrix}$; that is, $a_{22}=3$, $\mathbf{a}_{2k}^T=[-6]$, $\mathbf{a}_{k2}=[-6]$, and $\mathbf{A}_{kk}=[3]$, which yields $u_{22}=3$, $\mathbf{u}_{2k}^T=[-6]$, $\mathbf{l}_{k2}=[-2]$, and $\mathbf{D}_{kk}=[-9]$. In matrix form,

$$\begin{bmatrix} 2 & 1 & -2 \\ -0.5 & 3 & -6 \\ 1.5 & -6 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 1 & -2 \\ -0.5 & 3 & -6 \\ 1.5 & -2 & -9 \end{bmatrix}$$

$i=3$: At this point, since a 1×1 matrix remains (i.e., $[-9]$), the process is complete. The $\mathbf{L}$ and $\mathbf{U}$ matrices are, respectively,

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 1.5 & -2 & 1 \end{bmatrix}, \qquad \mathbf{U} = \begin{bmatrix} 2 & 1 & -2 \\ 0 & 3 & -6 \\ 0 & 0 & -9 \end{bmatrix}$$

The solution of $\mathbf{Ax} = \mathbf{b}$ can be obtained first by solving $\mathbf{Ly} = \mathbf{b}$ for y_1, then y_2, and then y_3; that is,

$$\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 1.5 & -2 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -9.5 \\ 10.5 \end{bmatrix} \rightarrow \mathbf{y} = \begin{bmatrix} 1 \\ -9 \\ -9 \end{bmatrix}$$

Solving $\mathbf{Ux} = \mathbf{y}$ for x_3, then x_2, and then x_1,

$$\begin{bmatrix} 2 & 1 & -2 \\ 0 & 3 & -6 \\ 0 & 0 & -9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -9 \\ -9 \end{bmatrix} \rightarrow \mathbf{x} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

4.5 NEWTON–RAPHSON METHOD FOR NONLINEAR EQUATIONS

One of the most frequently occurring problems in kinematics is to find solutions of nonlinear algebraic equations of the form

$$\mathbf{\Phi}(\mathbf{q}) = \mathbf{0} \tag{4.5.1}$$

When $\mathbf{\Phi}(\mathbf{q})$ and its Jacobian $\mathbf{\Phi}_{\mathbf{q}}$ can be evaluated, the iterative *Newton–Raphson method* may be used to find a solution of Eq. 4.5.1.

Newton–Raphson Method for One Equation in One Unknown Consider the single equation

$$\Phi(q) = 0 \tag{4.5.2}$$

that is nonlinear in the scalar variable q. Let $q = q^*$ be a solution of Eq. 4.5.1 and let $q^{(i)}$ be an approximation of q^*. The Taylor series expansion of $\Phi(q)$ about the point $q = q^{(i)}$ is [25, 26]

$$\Phi(q) = \Phi(q^{(i)}) + \Phi_q(q^{(i)})(q - q^{(i)}) + \text{higher-order terms} \tag{4.5.3}$$

Let $q = q^{(i+1)}$ be an improved approximation that is to be determined from the condition

$$\Phi(q^{(i+1)}) \approx \Phi(q^{(i)}) + \Phi_q(q^{(i)})(q^{(i+1)} - q^{(i)}) = 0 \tag{4.5.4}$$

where, if $q^{(i+1)} - q^{(i)}$ is small, higher-order terms are neglected. If $\Phi_q(q^{(i)}) \neq 0$, Eq. 4.5.4 can be solved to obtain

$$q^{(i+1)} = \frac{q^{(i)} - \Phi(q^{(i)})}{\Phi_q(q^{(i)})} \tag{4.5.5}$$

Equation 4.5.5 defines the *Newton–Raphson algorithm* for scalar equations that can be employed iteratively to produce a sequence of approximate solutions, beginning with an initial estimate of the solution of Eq. 4.5.2.

1. Make an estimate $q^{(0)}$ of the solution of Eq. 4.5.2.
2. In iterations $i = 0, 1, \ldots$, evaluate $\Phi(q^{(i)})$ and $\Phi_q(q^{(i)})$. If $|\Phi(q^{(i)})| < \varepsilon_e$, where ε_e is the equation error tolerance, and $|q^{(i)} - q^{(i-1)}| < \varepsilon_s$, where ε_s is the solution error tolerance, terminate. If $\Phi_q(q^{(i)}) = 0$ and $\Phi(q^{(i)}) \neq 0$, return to step 1 with a new estimate. Otherwise, go to step 3.
3. Calculate $q^{(i+1)}$ from Eq. 4.5.5 and return to step 2, with $i+1$ in place of i.

The sequence of approximate solutions generated by the Newton–Raphson algorithm, in many problems, will approach a solution of $\Phi(q) = 0$. However, if $q^{(0)}$ is not adequately close to q^*, the sequence may diverge. If the algorithm converges and the Jacobian is nonsingular, it is *quadratically convergent* [31]; that is, there is a constant c such that

$$|q^{(k+1)} - q^*| < c\,|q^{(k)} - q^*|^2 \tag{4.5.6}$$

As an illustration of this behavior, see Example 4.3.1. As might be expected, if $|q^{(0)} - q^*|$ is large, the error may grow quadratically.

The geometry of Newton–Raphson iteration is illustrated in Fig. 4.5.1 for a case in which it converges. The illustration in Fig. 4.5.2 shows that the method may diverge when the solution is an inflection point. If there are multiple solutions, the solution obtained is a function of the initial estimate, as shown in Fig. 4.5.3. If the initial estimate is near a local minimum or maximum, as shown in Fig. 4.5.4, the algorithm may fail to converge. It is essential that a reasonable estimate of the solution be used to initiate the iterative process.

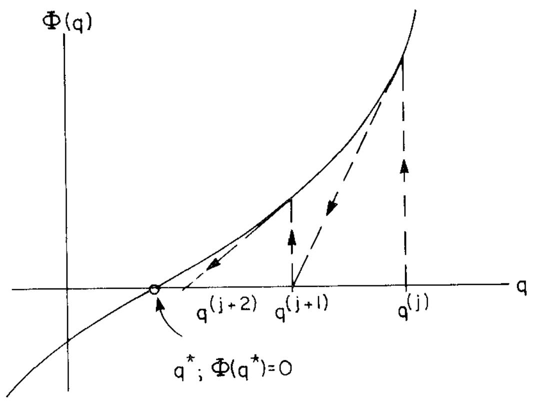


Figure 4.5.1 Graphic representation of Newton–Raphson method.

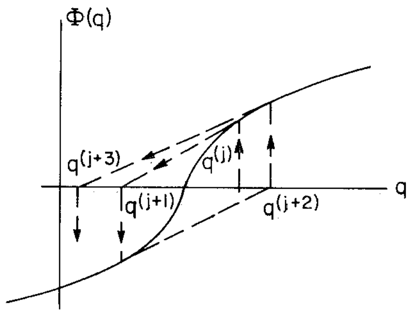


Figure 4.5.2 Root at inflection point.

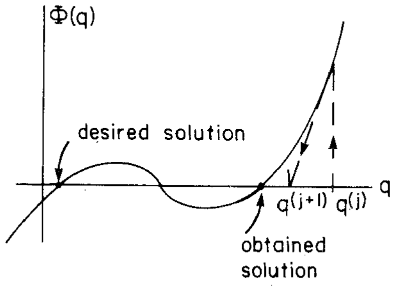


Figure 4.5.3 Multiple roots.

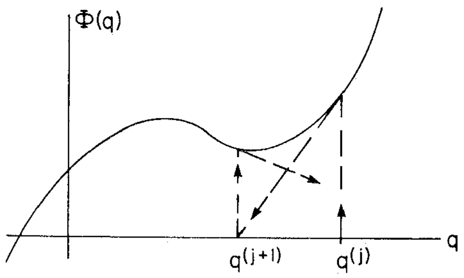


Figure 4.5.4 Divergence near a local minimum or maximum.

Newton–Raphson Method for *n* Equations in *n* Unknowns. Consider n nonlinear algebraic equations in n unknowns,

$$\begin{aligned} \Phi_1(\mathbf{q}) &\equiv \Phi_1(q_1, q_2, \ldots, q_n) = 0 \\ \Phi_2(\mathbf{q}) &\equiv \Phi_2(q_1, q_2, \ldots, q_n) = 0 \\ &\vdots \\ \Phi_n(\mathbf{q}) &\equiv \Phi_n(q_1, q_2, \ldots, q_n) = 0 \end{aligned} \tag{4.5.7}$$

where a solution vector is denoted as $\mathbf{q}^* \equiv [q_1^*, q_2^*, \ldots, q_n^*]^T$. A first-order Taylor expansion about an estimate $\mathbf{q}^{(i)}$ of the solution can be applied to Eqs. 4.5.7. Neglecting higher-order terms, the first-order Taylor approximation [25, 26] for an improved approximation $\mathbf{q}^{(i+1)}$ is

$$\mathbf{\Phi}(\mathbf{q}^{(i)}) + \mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}^{(i)})[\mathbf{q}^{(i+1)} - \mathbf{q}^{(i)}] = \mathbf{0}$$

If the Jacobian matrix $\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}^{(i)})$ is nonsingular, this equation may be solved for $\mathbf{q}^{(i+1)}$ by solving

$$\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}^{(i)})\Delta\mathbf{q}^{(i)} = -\mathbf{\Phi}(\mathbf{q}^{(i)}) \tag{4.5.8}$$

and setting $\mathbf{q}^{(i+1)} = \mathbf{q}^{(i)} + \Delta\mathbf{q}^{(i)}$.

The *Newton–Raphson algorithm* for systems of equations is as follows:

1. Make an initial estimate $\mathbf{q}^{(0)}$ of the solution of Eq. 4.5.7.
2. In iteration $i = 0, 1, 2, \ldots$, evaluate $\mathbf{\Phi}(\mathbf{q}^{(i)})$ and $\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}^{(i)})$. If the magnitudes of all errors and changes in approximate solutions satisfy

$$\left.\begin{aligned} |\Phi_k(\mathbf{q}^{(i)})| &\le \varepsilon_e \\ |q_k^{(i)} - q_k^{(i-1)}| &\le \varepsilon_s \end{aligned}\right\} \quad k = 1, \ldots, n$$

   terminate. If $\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}^{(i)})$ is singular and $\mathbf{\Phi}(\mathbf{q}^{(i)}) \neq \mathbf{0}$, return to step 1 and restart the process with a new estimate. Otherwise, go to step 3. Here, ε_e is the equation error tolerance and ε_s is the solution error tolerance.
3. Solve $\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}^{(i)})\Delta\mathbf{q}^{(i)} = -\mathbf{\Phi}(\mathbf{q}^{(i)})$, set $\mathbf{q}^{(i+1)} = \mathbf{q}^{(i)} + \Delta\mathbf{q}^{(i)}$, and return to step 2 with $i + 1$ in place of i.

Convergence properties of this algorithm for systems of equations are essentially the same as for scalar equations [31].

Numerical examples of the application of the Newton–Raphson method for kinematic position analysis have been presented, for two different mechanisms, in Examples 3.6.2 and 4.3.1.

4.6 DETECTION AND ELIMINATION OF REDUNDANT CONSTRAINTS

If redundant constraint equations exist in the nh kinematic constraint equations

$$\mathbf{\Phi}^K(\mathbf{q}) = \mathbf{0} \tag{4.6.1}$$

the number of system degrees of freedom is not $3nb - nh$. For a simple mechanical system, careful examination of the system may determine the number of degrees of freedom. However, this is not easy for even a moderately complicated system, so the correct number of independent constraint equations is often difficult to define.

Assume that a system is in an assembled configuration obtained by the minimization procedure of Section 4.3. If the constraint Jacobian of Eq. 4.6.1 has full row rank, then an $nh \times nh$ submatrix associated with dependent variables is nonsingular and, by the implicit function theorem, these dependent variables can be expressed in terms of the remaining $3nb - nh$ independent variables. To determine the row rank of a matrix, Gaussian elimination of Section 4.4 can be used with either row or full pivoting. Row and full pivoting theoretically give the same rank of the matrix, but full pivoting is numerically more reliable.

As physical motivation for the use of Gaussian elimination, consider a *virtual displacement* $\delta\mathbf{q}$ (a small variation in $\mathbf{q}$ with time held fixed) that is consistent with the kinematic constraint equations of Eq. 4.6.1; that is, it satisfies the linearized constraint equations,

$$\mathbf{\Phi}_{\mathbf{q}}^{K}\delta\mathbf{q} = \mathbf{0} \tag{4.6.2}$$

Gaussian elimination can be applied to the Jacobian matrix on the left of Eq. 4.6.2. The resulting form of the modified Jacobian matrix and equations is

$$\begin{bmatrix} \mathbf{\Phi}_{\mathbf{u}}^{KI} & \mathbf{\Phi}_{\mathbf{v}}^{KI} \\ \mathbf{0} & \mathbf{\Phi}_{\mathbf{v}}^{KR} \end{bmatrix} \begin{bmatrix} \delta\mathbf{u} \\ \delta\mathbf{v} \end{bmatrix} = \mathbf{0} \tag{4.6.3}$$

where $\mathbf{\Phi}_{\mathbf{u}}^{KI}$ is a triangular $nh' \times nh'$ matrix with ones in its diagonal, $nh' \leq nh$, and $\mathbf{\Phi}_{\mathbf{v}}^{KR}$ is a zero matrix, to within computation tolerance. If Gaussian elimination of Eq. 4.6.2 is done with finite digit arithmetic, errors are involved during the elimination procedure, and all elements of $\mathbf{\Phi}_{\mathbf{v}}^{KR}$ will not be exactly zero. It is thus necessary to set elements that are close to machine precision zero to exactly zero during the elimination procedure. Equation 4.6.2 is thus reduced to the independent equations

$$\mathbf{\Phi}_{\mathbf{u}}^{KI}\delta\mathbf{u} + \mathbf{\Phi}_{\mathbf{v}}^{KI}\delta\mathbf{v} = \mathbf{0} \tag{4.6.4}$$

Since $|\mathbf{\Phi}_{\mathbf{u}}^{KI}| = 1$, $\delta\mathbf{u}$ can be expressed as

$$\delta\mathbf{u} = -(\mathbf{\Phi}_{\mathbf{u}}^{KI})^{-1}\mathbf{\Phi}_{\mathbf{v}}^{KI}\delta\mathbf{v}$$

By the implicit function theorem, nh' constraint equations corresponding to $\mathbf{\Phi}_{\mathbf{u}}^{KI}$ are independent, that is,

$$\mathbf{\Phi}^{KI}(\mathbf{q}) = \mathbf{0} \tag{4.6.5}$$

and the remaining constraint equations $\mathbf{\Phi}^{KR}$ are redundant. If there exists an isolated singular point at the initial configurations, that is, a bifurcation point that has multiple solutions, an apparently redundant constraint can be detected. In this case, by perturbing a few generalized coordinates and reassembling the system with the fixed perturbed variables, if the Jacobian has increased row rank, it can be concluded that the constraint is redundant only at an isolated singular point. By checking redundant constraints at the reassembled configuration, the correct number of redundant constraints for the system can be determined. These dependent constraints are eliminated, yielding nh' independent kinematic constraints.

The system has $3nb - nh'$ degrees of freedom; so, for kinematic analysis, $3nb - nh' = d$ driving constraint equations

$$\mathbf{\Phi}^D(\mathbf{q}, t) = \mathbf{0} \tag{4.6.6}$$

must be independent of each other and must also be independent of the retained kinematic constraints of Eq. 4.6.1. If there exists a redundant driving constraint, it can be dependent on other driving constraint equations or it could be dependent on the kinematic constraint equations. Since the reduced nh' kinematic constraint equations of Eq. 4.6.5 are independent, it is desired to determine and eliminate only redundant driving constraint equations from Eq. 4.6.6.

To detect redundant driving constraint equations, Gaussian elimination of the Jacobian matrix of the combined constraints

$$\mathbf{\Phi}(\mathbf{q}, t) = \begin{bmatrix} \mathbf{\Phi}^K(\mathbf{q}) \\ \mathbf{\Phi}^D(\mathbf{q}, t) \end{bmatrix} = \mathbf{0} \tag{4.6.7}$$

can be used. If full pivoting Gaussian elimination is used, then kinematic constraint equations might be selected as dependent. To prevent this, only column exchange is allowed during Gaussian elimination through the diagonal elements of $\mathbf{\Phi}^I_{\mathbf{u}'}$. Beyond this point in Gaussian elimination, full pivoting is allowed. The resulting matrix can be expressed as

$$\begin{bmatrix} \mathbf{\Phi}^I_{\mathbf{u}'} & \mathbf{\Phi}^I_{\mathbf{v}'} \\ \mathbf{0} & \mathbf{\Phi}^D_{\mathbf{v}'} \end{bmatrix} \tag{4.6.8}$$

where the upper triangular matrix $\mathbf{\Phi}^I_{\mathbf{u}'}$ has ones on its diagonal and the upper triangular matrix $\mathbf{\Phi}^D_{\mathbf{v}'}$ can have zero rows. Driving constraint equations corresponding to zero rows of $\mathbf{\Phi}^D_{\mathbf{v}'}$ are identified as redundant driving constraints. Redundant driving constraints must be removed and replaced by independent driving constraints prior to kinematic analysis.

The numerical procedure used to automate the *redundant constraint elimination algorithm* may be summarized as follows:

1. From initial estimates, assemble the system to minimize kinematic constraint equation violation, as in Section 4.3. If assembly fails, advise the user that the design may be infeasible or a better estimate of the assembled configuration is needed.
2. After successful assembly, evaluate the Jacobian matrix of the kinematic constraint equations.

3. Using full pivoting Gaussian elimination, determine the rank of the kinematic constraint Jacobian. During the elimination procedure, record row and column interchanges. The resulting matrix has the form of Eq. 4.6.3. Eliminate redundant kinematic constraint equations $\mathbf{\Phi}_{\mathbf{v}}^{KR}$ from the system.
4. From step 3, a good choice of independent generalized coordinates can be reported from the column indexes of submatrix $\mathbf{\Phi}_{\mathbf{v}}^{KI}$ in Eq. 4.6.3. Report this suggested choice of independent coordinates and the number of degrees of freedom to the user, to help in the selection of driving constraints.
5. Add driving constraint equations at the bottom of the independent kinematic constraint equations, as in Eq. 4.6.7. Perform Gaussian elimination, with column exchanges only through the rows of $\mathbf{\Phi}_{\mathbf{q}}^{K}$ and full pivoting beyond. Identify redundant driving constraints that correspond to zero rows in $\mathbf{\Phi}_{\mathbf{v}}^{D}$ in Eq. 4.6.8. Eliminate redundant driving constraint equations from the system.
6. If the number of independent kinematic and driving constraint equations remaining $(nh' + d')$ is less than the number nc of generalized coordinates, then write a message that $nc - nh' - d'$ additional driving constraints are required to proceed with kinematic analysis.

Example 4.6.1: Consider the four-bar parallelogram mechanism shown in Fig. 4.6.1, with a unit distance constraint between points A and B. Equation 4.6.1, with $\mathbf{q} = [\phi_1, x_2, y_2, \phi_2, \phi_3]^T$, is

$$\mathbf{\Phi}^K(\mathbf{q}) = \begin{bmatrix} x_2 - \cos\phi_1 - \cos\phi_2 \\ y_2 - \sin\phi_1 - \sin\phi_2 \\ x_2 - \cos\phi_3 - 1 \\ y_2 - \sin\phi_3 \\ (x_2 + \cos\phi_2 - 2)^2 + (y_2 + \sin\phi_2)^2 - 1 \end{bmatrix} = \mathbf{0} \tag{4.6.9}$$

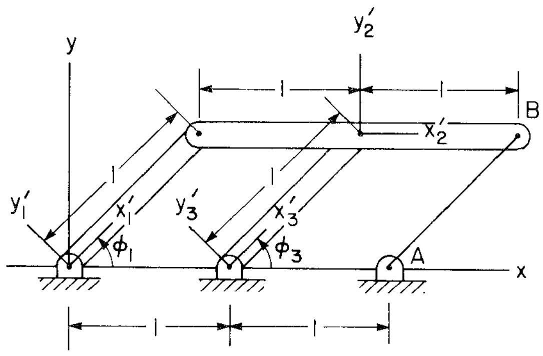


Figure 4.6.1 Four-bar crank mechanism with a redundant constraint.

As is geometrically clear, $\phi_1 = \phi_3$ and $\phi_2 = 0$. Thus, from the first and second of Eqs. 4.6.9, $x_2 = 1 + \cos\phi_1$ and $y_2 = \sin\phi_1$. Substituting these results into the last of Eqs. 4.6.9 shows that it is identically satisfied; that is, it is redundant.

To see that the constraint Jacobian row rank criterion identifies this redundancy, the Jacobian of Eq. 4.6.1 is evaluated at the assembled configuration as

$$\mathbf{\Phi}^K_{\mathbf{q}}(\mathbf{q}) = \begin{bmatrix} \sin\phi_1 & 1 & 0 & \sin\phi_2 & 0 \\ -\cos\phi_1 & 0 & 1 & -\cos\phi_2 & 0 \\ 0 & 1 & 0 & 0 & \sin\phi_3 \\ 0 & 0 & 1 & 0 & -\cos\phi_3 \\ 0 & 2(x_2+\cos\phi_2-2) & 2(y_2+\sin\phi_2) & \begin{matrix} -2\sin\phi_2(x_2+\cos\phi_2-2) \\ +2\cos\phi_2(y_2+\sin\phi_2) \end{matrix} & 0 \end{bmatrix} \tag{4.6.10}$$

Let $\phi_1^0 = \phi_3^0 = 1.0472$ (60°) and $\phi_2^0 = 0$, so $x_2 = 1.5$ and $y_2 = 0.866$, and evaluate the Jacobian of Eq. 4.6.10. After Gaussian elimination, without pivoting on the last row, the Jacobian is reduced to

$$\begin{bmatrix} 1 & 0 & 0 & 0.866 & 0 \\ 0 & 1 & -1 & -0.5 & 0 \\ 0 & 0 & 1 & 0.5 & -0.5 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

During the elimination, the last row did not change its position; that is, the reduced Jacobian matrix is rank deficient and the last constraint is identified as a redundant constraint. Even if a small change is made in ϕ_1^0, and hence ϕ_3^0, the same conclusion follows, so the redundancy is not an isolated singular point.

As a second check, consider the Jacobian of Eq. 4.6.10 at $\phi_1^0 = \phi_3^0 = 0$ and $\phi_2^0 = 0$. After Gaussian elimination, using full pivoting, the Jacobian is reduced to

$$\begin{bmatrix} 1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which has a row rank deficiency of 2. This suggests that two constraints may be redundant. However, an arbitrarily small perturbation of ϕ_1^0, and hence ϕ_3^0, yields a matrix with row rank deficiency of only 1. Thus, in addition to a redundant constraint, at $\phi_1^0 = \phi_3^0 = 0$ the mechanism has an isolated singular point.

PROBLEMS

Section 4.2

4.2.1. Write the explicit form of the kinematic and driving constraints of Example 4.2.1.

4.2.2. Verify that the Jacobian matrix of Eq. 4.2.6 is correct by direct differentiation of the constraints of Prob. 4.2.1.

4.2.3. Evaluate γ for Example 4.2.1.

4.2.4. Write the position, velocity, and acceleration equations for the simple pendulum shown in Fig. P4.2.4, with angle driver ωt and

(a) modeling ground as body 2, with absolute constraints, and imposing a revolute constraint between ground and the pendulum (body 1).

(b) modeling the pivot between the global frame and pendulum using absolute x and y constraints (i.e., a one-body model).

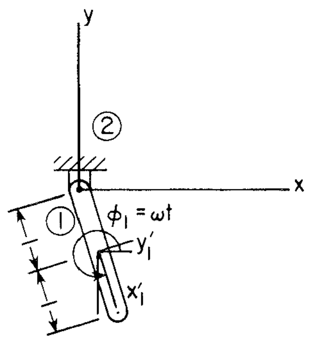


Figure P4.2.4

Section 4.4

4.4.1. Use Gaussian elimination without pivoting to solve the equations

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$$

4.4.2. Solve Prob. 4.4.1 using Gaussian elimination with row pivoting.

4.4.3. Solve Prob. 4.4.1 using Gaussian elimination with full pivoting.

4.4.4. Repeat the forward elimination of Example 4.4.4 with a right side $\mathbf{b} = [3, 2, 5]^T$ and determine whether a solution exists.

4.4.5. Repeat the forward elimination of Example 4.4.4 with full pivoting. What dependent–independent variable partitioning is obtained?

4.4.6. Apply **L–U** factorization to matrix **A** and solve the set of algebraic equations

$\mathbf{Ax} = \mathbf{b}$ for the unknown $\mathbf{x}$, where

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 1 \\ 2 & 1 & 0 & 2 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 0 \end{bmatrix}$$

Section 4.5

4.5.1. Use the Newton–Raphson method to solve $\Phi(q) = (q-1)^{1/3} = 0$, with $q^{(0)} = 2$. Confirm the solution behavior of Fig. 4.5.2.

4.5.2. Use the Newton–Raphson method to solve $\Phi(q) = q^3 - 4q + 4 = 0$, with $q^{(0)} = 1.5$. Confirm the solution behavior of Fig. 4.5.4.

4.5.3. Consider the four-bar mechanism of Example 3.5.2, with $\omega = \pi$ rad/s. Solve the constraint equation of Eq. 4.5.1 at $t = \frac{1}{3}$ s using the Newton–Raphson method with $\mathbf{q}^{(0)} = [\pi/3,\ 5.0,\ 4.0,\ 0.2,\ 0.9]^T$ and $\varepsilon_e = \varepsilon_s = 10^{-5}$. Monitor $\|\mathbf{q}^{(i)} - \mathbf{q}^*\|$, where $\mathbf{q}^* = [\pi/3,\ 4.94949,\ 4.01222,\ 0.18376,\ 0.86344]^T$ is the exact solution, and confirm that the Newton–Raphson method converges quadratically.

Section 4.6

4.6.1. Consider the one degree of freedom double pendulum of Example 3.6.2. If, instead of the constraint equations of Example 3.6.2, the kinematic constraint equations

$$\mathbf{\Phi}^K(\mathbf{q}) = \begin{bmatrix} x_1 - \cos\phi_1 \\ y_1 - \sin\phi_1 \\ x_2 - \cos\phi_1 - \cos\phi_2 \\ x_2 - x_1 - \cos\phi_2 \\ y_2 + 1 \end{bmatrix} = \mathbf{0}$$

are used, determine if there is a redundant constraint by evaluating the Jacobian and performing Gaussian elimination at $\phi_1 = 5\pi/3$ and $\phi_2 = 6.14881$ rad. By monitoring row index, decide which constraint is redundant.