MATLAB PROGRAMS
for Textbook

Kinematics, Dynamics, and Design of Machinery

by
K. J. Waldron and G. L. Kinzel

Department of Mechanical Engineering

THE OHIO STATE UNIVERSITY

Table of Contents

Section **Page No.**

1.0 Overview 1
1.1 Programs Available 1
1.2 Running MATLAB 4
1.2.1 Running Programs on a Macintosh 4
1.2.2 Running Programs Under Microsoft Windows on a PC 4
1.2.3 MATLAB Graphics Window 5
1.2.4 Help in Using MATLAB 5

2.0 Programs for Chapter 2 6
2.A.1 MATLAB Routine for The Centrodes of a Four-Bar Mechanism 6
2.A.2 Line Intersection Routine (lineintersect.m) 6
2.A.3 Locating C3 Relative to Coupler 7
2.A.4 Centrode Branches 7
2.A.5 Centrode Drawing Routine (centrodes.m) 8
2.A.6 Sample run using centrodes.m 8

3.0 Programs for Chapter 3:Mechanism Analysis 10
3.A MATLAB Function For Four-Bar Linkage Analysis 10
3.A.1 Four-Bar Analysis Routine when the Crank Is the Driver 10
3.A.1.1 Source code for sample m-file (Examples 3.1 and 3.2) using fourbar_cr.m 11
3.A.2 Four-Bar Analysis Routine when the Coupler Is the Driver 12
3.A.2.1 Source code for sample m-file (Examples 3.1 and 3.2) using fourbar_co.m 12
3.A.3 MATLAB Procedure for Analyzing a Four-Bar Mechanism for a Complete Cycle of Motion 13
3.A.3.1 Overview 13
3.A.3.2 Angle Limits for Crank (r2) as Input (fb_angle_limits_cr.m) 13
3.A.3.2.1 Condition When $r_1 + r_2 \le r_3 + r_4$ and $|r_1 - r_2| \ge |r_3 - r_4|$ 13
3.A.3.2.2 Condition When $r_1 + r_2 \ge r_3 + r_4$ and $|r_1 - r_2| \ge |r_3 - r_4|$ 14
3.A.3.2.3 Condition When $r_1 + r_2 \ge r_3 + r_4$ and $|r_1 - r_2| \le |r_3 - r_4|$ 14
3.A.3.2.4 Condition When $r_1 + r_2 \le r_3 + r_4$ and $|r_1 - r_2| \le |r_3 - r_4|$ 14
3.A.3.3 Angle Limits for Coupler (r3) as Input (fb_angle_limits_co.m) 15
3.A.3.3.1 Condition When $r_1 + r_3 \le r_2 + r_4$ and $|r_1 - r_3| \ge |r_2 - r_4|$ 15
3.A.3.3.2 Condition When $r_1 + r_3 \ge r_2 + r_4$ and $|r_1 - r_3| \ge |r_2 - r_4|$ 15
3.A.3.3.3 Condition When $r_1 + r_3 \ge r_2 + r_4$ and $|r_1 - r_3| \le |r_2 - r_4|$ 15
3.A.3.3.4 Condition When $r_1 + r_3 \le r_2 + r_4$ and $|r_1 - r_3| \le |r_2 - r_4|$ 15
3.A.3.4 Analysis Program(fourbar_analysis.m) 16
3.A.3.5 Sample run using fourbar_analysis.m 17
3.B MATLAB Function For Rigid Body Analysis 19
3.B.1 Rigid Body Routine When A and B Are Known (rbody1.m) 19
3.B.1.1 Source Code for Sample m-file (Examples 3.3) Using rbody1.m 20
3.B.2 Rigid Body Routine When A and B Are Known (rbody2.m) 20
3.B.2.1 Source Code for Sample m-file (Example 3.3) Using rbody2.m 21
3.C MATLAB Functions For Slider-Crank Analysis 22
3.C.1 Slider-Crank Routine when the Crank Is Driver (sldcrkc.m) 22
3.C.1.1 Source Code for Sample m-file (Example 3.4) Using sldcrkc.m 23
3.C.2 Slider-Crank Routine when the Coupler Is the Driver (sldcrkco.m) 23

Section **Page No.**

3.C.2.1 Source Code for Sample m-file (Example 3.4) Using sldcrkco.m....24
3.C.3 Slider-Crank Routine when the Slider Is the Driver (sldcrks.m)....25
3.C.3.1 Source Code for Sample m-file (Examples 3.5) Using sldcrks.m....25
3.C.4 MATLAB Procedure for Analyzing Slider-Crank Mechanism for a Complete Cycle (slidercrank_anal.m)....26
3.C.4.1 Overview....26
3.C.4.2 Angle Limits for Crank (r2) as Input (sc_angle_limits_cr.m)....26
3.C.4.2.1 Mechanism When $0 \le r4 \le r3$ and $r3 + r4 \le r2$....28
3.C.4.2.2 Mechanism When $0 \le r3 \le r4$ and $r3 + r4 \ge r2$....28
3.C.4.2.3 Mechanism When $0 \le -r4 \le r3$ and $r3 - r4 \le r2$....28
3.C.4.2.4 Mechanism When $0 \le r3 \le -r4$ and $r3 - r4 \ge r2$....28
3.C.4.3 Angle Limits for the Coupler (r3) as Input (sc_angle_limits_co.m)....28
3.C.4.3.1 Mechanism When $0 \le r4$ and $r2 + r4 \ge r3$....28
3.C.4.3.2 Mechanism When $0 \le r4$ and $r2 + r4 \le r3$....29
3.C.4.3.3 Mechanism When $0 \ge r4$ and $r2 + r4 < r3$....30
3.C.4.3.4 Mechanism When $0 \ge r4$ and $r2 - r4 \le r3$....30
3.C.4.4 Limits when the Slider (r1) as Input (sc_angle_limits_sl.m)....30
3.C.4.4.1 Mechanism When $|r2 - r3| > r4$....30
3.C.4.4.2 Mechanism When $|r2 - r3| < r4$....31
3.C.4.5 Slider-Crank Analysis Routine slidercrank_anal.m....31
3.C.4.6 Sample run using slidercrank_anal.m....31
3.D MATLAB Functions For Inverted Slider-Crank Analysis....34
3.D.1 Slider-Crank Routine when the Crank Is Driver (isldcrkc.m)....34
3.D.1.1 Source Code for Sample m-file (Examples 3.6) Using isldcrkc.m....35
3.D.2 Slider-Crank Routine when the Coupler Angle Is Driver (isldcrkco.m)....35
3.D.2.1 Source Code for Sample m-file (Examples 3.6) Using isldcrkco.m....36
3.D.3 Inverted Slider-Crank Routine when the Slider Is Driver (isldcrks.m)....36
3.D.3.1 Source Code for Sample m-file (Example 3.6 with R3 as input) using isldcrks.m....37
3.E MATLAB Functions For RPRP Mechanism Analysis....37
3.E.1 RPRP Mechanism Analysis Routine when the Crank Is Driver (rprpc.m)....38
3.E.1.1 Source Code for Sample m-file (Example 3.7) using rprpc.m....39
3.E.2 RPRP Mechanism Analysis Routine When the Slider on the Frame Is the Driver (rprps.m)39
3.E.2.1 Source Code for Sample m-file (Example 3.7 with R4 as Input) Using rprps.m....40
3.E.3 RPRP Mechanism Analysis Routine When the Slider on the Coupler Is the Driver (rprpsc.m)....40
3.E.3.1 Source Code for Sample m-file (Example 3.7 with R3 as Input) Using rprpsc.m....41
3.F MATLAB Functions For RRPP Mechanism Analysis....42
3.F.1 RRPP Mechanism Analysis Routine when the Crank Is Driver (rrppc.m)....42
3.F.1.1 Source Code for Sample m-file (Example 3.8) Using rrppc.m....43
3.F.2 RRPP Mechanism Analysis Routine when the Slider on the Frame Is the Driver (rrpps.m) .43
3.F.2.1 Source Code for Sample m-file (Example 3.8 with R1 as input) Using rrpps.m....44
3.F.3 RRPP Mechanism Analysis Routine when the Slider on the Coupler Is the Driver (rrppsc.m)....44
3.F.3.1 Source Code for Sample m-file (Example 3.8 with R1 as input) Using rrpps.m....45
3.G MATLAB Functions For Elliptic Trammel Analysis....45
3.G.1 RRPP Mechanism Analysis Routine when the Crank Is Driver (prrpc.m)....46
3.G.1.1 Source Code for Sample m-file (Example 3.9) using rpprc.m. 47
3.G.2 PRRP Mechanism Analysis Routine when the Slider at Link 2 Is the Driver (prrps.m) 51
3.G.2.1 Source Code for Sample m-file (Example 3.9 with R1 as the Input) using rpprs.m....52
3.H MATLAB Utility Routines For Rectangle, Frameline, Circle, and Bushing 52
3.H.1 Rectangle Routine....52
3.H.2 Frameline Routine....53
3.H.3 Circle Routine....53

Section **Page No.**

3.H.4 Bushing Routine 54
3.I MATLAB Functions For RPPR Mechanism Analysis 55
3.I.1 RPPR Mechanism Analysis Routine when the Crank Is Driver (rpprc.m) 55
3.I.1.1 Source Code for Sample m-file (Example 3.10) Using rpprc.m 56
3.I.2 RPPR Mechanism Analysis Routine when the Slider on the Crank Is the Driver (rpprs.m) ... 56
3.I.2.1 Source Code for Sample m-file (Example 3.10 with R2 as input) Using rpprs.m 57

4.0 Programs for Chapter 4: Linkage Design 58
4.A MATLAB Functions for Rigid Body Guidance 58
4.A.1 Design Routines for Rigid Body Guidance using a Four-Bar Linkage 58
4.A.1.1 Assembly Mode Routine (assemblymode.m) 59
4.A.1.2 Center Point Routine (centerpoint.m) 59
4.A.1.3 Circle Point Routine (circlepoint.m) 59
4.A.1.4 Circle of Sliders Routine (cosline.m) 60
4.A.1.5 Fourbar Mechanism Analysys Routine (fourbar.m) 60
4.A.1.6 Pole Routine (pole.m) 61
4.A.1.7 Image Pole Routine (ipole.m) 61
4.A.1.8 Sample Run using rbg_4bar.m 62
4.A.2 Design Routines for Rigid Body Guidance using a Slider-Crank Linkage 64
4.A.2.1 Assembly Mode Routine (assemblymode_sc.m) 64
4.A.2.2 Slider Point Routine (sliderpoint.m) 64
4.A.2.3 Rectangle Routine (rectangle.m) 65
4.A.2.4 Slider-Crank Routine (sldcrks.m) 65
4.A.2.5 Frame Line Routine (frameline.m) 66
4.A.2.6 Sample Run using rbg_slidercrank.m 66
4.A.3 Design Routines for Rigid Body Guidance using a Crank-Slider Linkage 67
4.A.3.1 Assembly Mode Routine (assemblymode_cs.m) 67
4.A.3.2 Crank-Slider Routine (sldcrkc.m) 69
4.A.3.3 Sample Run using rbg_crankslider.m 70
4.A.4 Design Routines for Rigid Body Guidance using an Elliptic Trammel Linkage 72
4.A.4.1 Sample Run using rbg_el_trammel.m 72
4.B MATLAB Procedure for Function Generation 74
4.B.1 Overview 75
4.B.2 Linkage Classification 75
4.B.2.1 Case when $r_2 = r_{min}$ 75
4.B.2.2 Case when $r_3 = r_{min}$ *or* $r_4 = r_{min}$ 75
4.B.3 Function Routines 76
4.B.3.1 Animation Routine (ani.m) 76
4.B.3.2 Calculation Routine (animat.m) 77
4.B.3.3 Precision point calculation routine (chebychev.m) 77
4.B.3.4 Error routine (errors.m) 78
4.B.3.5 Function routine (funct.m) 78
4.B.3.6 Link routine (links.m) 78
4.B.3.7 Output value routine (phi2y.m) 79
4.B.3.8 Output value routine (precout.m) 79
4.B.4 Sample results 80
4.C MATLAB Procedure for Crank Rocker Design 84
4.C.1 Overview 84
4.C.2 Results from Sample Run of Crank-Rocker Synthesis 84
4.D MATLAB Procedure for Generating Coupler Curves 86
4.D.1 Overview 87
4.D.2 Crank-rocker Routine (hr_crankrocker.m) 87

Section **Page No.**

4.D.2.1 Sample run using hr_crankrocker.m 87
4.D.2.2 MATLAB Routine for Crank-Rocker Coupler Curve Program 90
4.D.3 Slider-Crank Routine (hr_slidercrank.m) 90
4.D.3.1 Sample run using hr_slidercrank.m 91
4.D.3.2 MATLAB Routine for Slider-Crank Coupler Curve Program 94
4.E MATLAB Procedure for Crank and Dyad Analysis 94
4.E.1 Overview 94
4.E.2 Crank Analysis 94
4.E.3 MATLAB Routine for Computing Crank Variables 95
4.E.4 Dyad Analysis 95
4.E.5 MATLAB Routine for Solving Dyad Equations 96
4.F MATLAB Procedure for 6-Link Dwell Mechanism Analysis 96
4.F.1 Introduction 96
4.F.2 Sample Run Using sixbar.m 98
4.G MATLAB Procedure for Generating Cognate Linkages 100
4.G.1 Introduction 100
4.G.2 Sample Run Using cognates.m 101
4.H MATLAB Procedure for Euler-Savary Equation 103
4.H.1 Introduction 103
4.H.2 Two Infinitesimally Separated Positions 103
4.H.3 Three Infinitesimally Separated Positions 106
4.H.3.1 Center of Curvature of Path of Moving Point Relative to Frame 106
4.H.3.2 Synthesis Using the Center of Curvature 108
4.H.3.3 Inflection Circle 109
4.H.3.4 Different Forms for the Euler-Savary Equation 112
4.H.4 Relationship Among IC, Centrodes, IC Tangent, and IC Velocity 119
4.H.5 Analytical Form for Euler Savary Equation 120
4.H.6 The Bobillier Constructions 120
4.H.6.1 Bobillier's Theorem 120
4.H.6.2 First Bobillier Construction 122
4.H.6.3 Second Bobillier Construction 123
4.H.7 MATLAB Routines for Inflection-Circle Calculations 125
4.H.7.1 First Inflection Circle Calculation Routine (inflection1.m) 126
4.H.7.2 Second Inflection Circle Calculation Routine (inflection2.m) 126
4.H.7.3 Third Inflection Circle Calculation Routine (inflection3.m) 127
4.H.7.4 Inflection Circle Routine (inflection_circle.m) 128
4.H.7.5 Sample run using inflection_circle.m 128
4.H.7.6 Inflection Circle Routine for Four-Bar Linkage (inflection_4bar.m) 129
4.H.7.6.1 Sample run using inflection_4bar.m 129

5.0 Programs for Chapter 5 131

6.0 Programs for Chapter 6: MATLAB Programs for Cam Analysis 132
6.A MATLAB Procedure for Cam Design for Axial Cylindrical-Faced Follower 132
6.A.1 Overview 132
6.A.2 Matlab Routines for Roller Follower Cam System (rf_cam.m) 133
6.A.3 Follower Routine (follower.m) 133
6.A.4 Results from Sample Run of rf_cam (Example 6.7) 134
6.B MATLAB Procedure for Cam Design for Axial Flat-Faced Follower 136
6.B.1 Overview 136
6.B.2 Matlab Design Routines for Flat-Faced Follower Cam System (ff_cam.m) 137
6.C MATLAB Procedure for Cam Design for Oscillating, Cylindrical-Faced Follower 140
6.C.1 Overview 140

Section **Page No.**

6.C.2 Matlab Routines for Oscillating, Cylindrical-Faced Follower Cam System (orf_cam.m)....140
6.C.3 Follower Routine (o_follower.m)....141
6.C.4 Results from Sample Run of orf_cam (Example 6.9)....141
6.D MATLAB Procedure for Cam Design for Oscillating, Flat-Faced Follower....143
6.D.1 Overview....143
6.D.2 Matlab Design Routines for an Oscillating, Flat-Faced Follower Cam System (off_cam.m) 144
6.D.3 Follower Routine (o_follower2.m)....144
6.D.4 Results from Sample Run of off_cam (Example 6.10) 144

7.0 Programs for Chapter 7 147

8.0 Programs for Chapter 8: Gear Drawing and Analysis....148
8.A MATLAB Program for Finding the Inverse Involute 148
8.B MATLAB Program for Drawing Spur Gear Profile....148
8.B.1 Overview 148
8.B.2 Results from Sample Run of geardr.m....148
8.C MATLAB Program for Drawing Conjugate Tooth Form....150
8.C.1 General Conjugate Tooth Forms....150
8.C.1.1 Required Geometric Parameters....151
8.C.1.2 Determination of the Point of Contact....151
8.C.1.3 Coordinate Transformations....154
8.D MATLAB Program for Drawing Rack Envelope (rackmotion.m)....160
8.D.1 Overview....160
8.D.2 Results from Sample Run of rackmotion.m....160

9.0 Programs for Chapter 9 162

10.0 Programs for Chapter 10 162

11.0 Programs for Chapter 11 162

12.0 Programs for Chapter 12 162

13.0 Programs for Chapter 13: Slider-Crank Balancing 163
13.A MATLAB Program for Balancing Slider-Crank Mechanism....163

14.0 Brief Overview of MATLAB for Applications in Kinematics 166
14.1 Introduction 166
14.2 M-files....166
14.3 Manipulating Data with MATLAB 167
14.3.1 Pointwise operations with matrices 169
14.3.2 Matrixwise operations with matrices....170
14.4 Statements, expressions in MATLAB....170
14.4.1 General statements....170
14.4.2 Matrix building functions 171
14.4.3 Relational operators 172
14.4.4 If-statements 173
14.4.5 For-loops....173
14.4.6 While-loops....174
14.5 Functions in MATLAB....174
14.5.1 Scalar functions 174
14.5.2 Vector functions....174

14.5.3 Matrix functions 175

Section **Page No.**

14.5.4 Submatrices and colon notation 175
14.5.5 User defined functions 176
14.6 MATLAB utility commands 177
14.6.1 help 177
14.6.2 plot, subplot, semilogx, semilogy, loglog, grid, title, xlabel, ylabel 177
14.6.3 Text strings, error messages, input 180
14.7 Animation Graphics 180
14.7.1 Introduction 180
14.7.2 Identifying the Viewport and Plotting Axes 181
14.7.3 Scaling the Axes 182
14.7.4 Line Objects 182
14.7.5 Updating Line Objects 184
14.7.6 Input functions 185
14.7.7 Miscellaneous Drawing Commands 186
14.8 Summary of MATLAB Functions 187
14.8.1 Color (Color Control and Lighting Models) 187
14.8.2 Datafun (Data Analysis and Fourier Transformations) 187
14.8.3 Demos (Demonstration and Samples) 188
14.8.4 Elfun (Elementary Math Functions) 189
14.8.5 Elmat (Elementary Matrices and Manipulation) 190
14.8.6 Funfun (Function Functions) 190
14.8.7 General (General Purpose Command) 190
14.8.8 Graphics (General Purpose Graphics Functions) 191
14.8.9 Iofun (Low-Level File I/O Functions) 191
14.8.10 Lang (Language Constructs and Debuggings) 192
14.8.11 Matfun (Matrix Functions) 192
14.8.12 OPS (Operators and Special Characters) 192
14.8.13 Plotxy (Two-Dimensional Graphics) 193
14.8.14 Plotxyz (Three-Dimensional Graphics) 193
14.8.15 Polyfun (Polynomial and Interpolation Functions) 193
14.8.16 Sparfun (Sparce Matrix Functions) 194
14.8.17 Specfun (Specialized Math Function) 194
14.8.18 Specmat (Specialized Matrices) 194
14.8.19 Sounds (Sound Processing Functions) 194
14.8.20 Strfun (Character String Functions) 194

MATLAB PROGRAMS

to Supplement the Textbook

Kinematics, Dynamics, and Design of Machinery, First Edition

1.0 Overview

This supplement to the textbook entitled *Kinematics, Dynamics, and Design of Machinery* by K.J. Waldron and G. L. Kinzel gives a description of the MATLAB programs written to support the textbook. The programs are organized as appendices to the chapters in the book on which the programs are based. Not all of the chapters have programs associated with them. **The programs require version 5.0 or higher of MATLAB**. Either the full version or student version of MATLAB may be used.

In general, the description consists of a brief overview of the purpose of the program followed by a description of the m-files involved. The input information is next described and one or more samples of the output are then given. When possible, the sample outputs correspond to the examples given in the text.

1.1 Programs Available

The programs address the following types of problems.

Cam Design	MATLAB program for cam design with axial cylindrical-faced follower, axial flat-faced follower, oscillating cylindrical-faced follower, and oscillating flat-faced follower
Centrode Plot	MATLAB program for computing the centrode for a four-bar linkage
Cognate Drawing	MATLAB programs for computing the cognate linkages for a four-bar linkage
Coupler Curve Gen.	MATLAB programs for computing the coupler curves of a four-bar linkages and slider cranks
Crank Rocker Design	MATLAB program for crank rocker design
Elliptic Trammel	Elliptic trammel mechanism analysis with either a slider or the coupler as driver
Footpump Analysis	Files for analyzing a footpump mechanism
Four Bar Analysis	Linkage analysis routines for a four-bar with either a crank or the coupler as driver
Function Generation	Program for function generation
Gear Drawing	Programs for drawing gears given the geometry of the cutter
Inflection Circle Prog.	Program for computing the inflection circle of a four bar linkage
Inverted Slider Crank	Analysis of an inverted slider-crank mechanism with the slider, coupler,, or crank as driver
Oldham Analysis	Oldham mechanism analysis with either the slider or crank as driver

Rectangle & Frameline	Utility programs for computing the coordinates of a rectangle and frameline
Rigid Body Analysis	Analysis of a rigid body if two points are known
Rigid Body Guidance	Program for design of linkage for three positions for rigid body guidance
RPRP Analysis	RPRP mechanism analysis with either the slider or crank as driver
RRPP Analysis	RRPP mechanism analysis with either the slider or crank as driver
SC Shaking Force	Program for computing the shaking force for a slider-crank mechanism
Six Bar Mechanism	Program for analyzing a six-bar linkage
Slider Crank Analysis	Slider-crank linkage analysis with either the slider, coupler, or crank as driver

Some of the M-files call function routines and use data files. The routines called are identified by the extension .m in Table 1.1. The data files have the extension .dat.

Table 1.1: Tree structure for the kinematic m-files

Main Routine	Ch.	Functions Used			Data Files Used
arb2th.m	8	axisadjust.m			arb2th.dat
centrodes.m	2	axisadjust.m lineintersect.m	circle.m fourbar.m	bushing.m	draglink.dat centrode.dat
cognates.m	4	axisadjust.m fourbar.m	circle.m	bushing.m	cognate.dat paper.dat
crank_rocker_design.m	4	axisadjust.m fourbar.m	circle.m	bushing.m	
example_3p1.m	3	fourbar_cr.m			
example_3p1t.m	3	dyad.m	crank.m		
example_3p10.m	3	fourbar_cr.m			
example_3p10t.m	3	dyad.m	crank.m		
example_3p10_2.m	3	rpprs.m			
example_3p10_3.m	3	rpprc.m			
example_3p1_2.m	3	fourbar_co.m			
example_3p2.m	3				
example_3p3b1.m	3	rbody1.m			
example_3p3b2.m	3	rbody2.m			
example_3p4.m	3	sldcrkc.m			
example_3p4_2.m	3	sldcrkco.m			
example_3p4_3.m	3	sldcrkc.m			
example_3p5.m	3	sldcrks.m			
example_3p6.m	3	isldcrkc.m			
example_3p6_2.m	3	isldcrks.m			
example_3p6_3.m	3	isldcrkc.m	axisadjust.m		
example_3p6_4.m	3	isldcrkco.m			
example_3p6_5.m	3	isldcrkc.m			
example_3p7.m	3	rprpc.m			
example_3p7_2.m	3	rprps.m			
example_3p7_3.m	3	rprpsc.m			

Table 1.1: Tree structure for the kinematic m-files (continued)

Main Routine	Ch.	Functions Used			Data Files Used
example_3p8.m	3	rrppc.m			
example_3p8_2.m	3	rrpps.m			
example_3p8_3.m	3	rrppsc.m			
example_3p9.m	3	prrpc.m	rect.m	frameline.m	
example_3p9_2.m	3	prrps.m			
example_8p1	8				
example_8p2	8				
example_8p3	8	inverse_inv.m			
example_8p4	8	inverse_inv.m			
ff_cam.m	6	axisadjust.m rect.m	circle.m follower.m	bushing.m frameline.m	ff_camio.dat
fig_8p7	8	axisadjust.m			
footpump.m	3	axisadjust.m	circle.m	bushing.m	
fourbar_analysis.m	3	axisadjust.m fb_angle_limits_co.m fourbar_cr.m	circle.m fb_angle_limits_cr.m	bushing.m fourbar_co.m	fourbario.dat watt.dat cheby.dat
functgen.m	4	axisadjust.m animat.m fourbar.m links.m	circle.m errors.m chebychev.m precout.m	bushing.m phi2y.m funct.m ani.m	functgenio.dat
geardr.m	8	axisadjust.m			geardr.dat
hr_crankrocker.m	4	axisadjust.m rbody1.m	circle.m fourbar.m	bushing.m	hrcrankrockerio.dat
hr_slidercrank.m	4	axisadjust.m rect.m sldcrkc.m	circle.m frameline.m	bushing.m rbody1.m	hrslidercrankio.dat
inflection_4bar.m	4	inflection2.m pole.m bushing.m	inflection3.m axisadjust.m fourbar.m	circle.m lineintersect.m inflection1.m	inflection4bario.dat
inflection_circle.m	4	inflection3.m	axisadjust.m		inflectioncirio.dat
off_cam.m	6	axisadjust.m pole.m	circle.m o_follower2.m	bushing.m	off_camio.dat
orf_cam.m	6	axisadjust.m pole.m	circle.m o_follower.m	bushing.m	orf_camio.dat
rackmotion.m	8	axisadjust.m			rackmotion.dat
rbg_4bar.m	4	axisadjust.m circlepoint.m assemblymode.m fourbar.m	circle.m ipole.m centerpoint.m	bushing.m cosline.m pole.m	rbg_4bario.dat
rbg_crankslider.m	4	axisadjust.m centerpoint.m pole.m assemblymode_cs.m sldcrkc.m	circle.m circlepoint.m sliderpoint.m rect.m	bushing.m ipole.m cosline.m frameline.m	rbg_crksldio.dat
rbg_el_trammel.m	4	axisadjust.m cosline.m prrpc.m	circle.m ipole.m pole.m	sliderpoint.m frameline.m rect.m	rbg_eltrammelio.dat
rbg_slidercrank.m	4	axisadjust.m	circle.m	bushing.m	rbg_sldcrkio.dat
		centerpoint.m	circlepoint.m	ipole.m	
		pole.m	sliderpoint.m	cosline.m	
		assemblymode_sc.m	frameline.m	rect.m	
		sldcrks.m			

Table 1.1: Tree structure for the kinematic m-files (continued)

Main Routine	Ch.	Functions Used			Data Files Used
rf_cam.m	6	axisadjust.m frameline.m	circle.m pole.m	bushing.m follower.m	rf_camio.dat
shake.m	13	axisadjust.m rect.m sldcrkc.m	circle.m sc_angle_limits_cr.m	bushing.m frameline.m	shake.dat
sixbar.m	4	axisadjust.m dyad.m	circle.m rbody1.m	bushing.m fourbar.m	sixbario.dat figure8.dat
slidercrank_anal.m	3	axisadjust.m rect.m sldcrkco.m sldcrks.m	circle.m frameline.m sc_angle_limits_sl.m sc_angle_limits_co.m	bushing.m sc_angle_limits_cr.m sldcrkc.m	evans.dat slidercrkio.dat
Utility fundtion	8	inverse_inv			

1.2 Running MATLAB

Before running a given program (m-file), it is convenient to have the program, all of the functions it calls, and all of the data files used in the same directory. Therefore, on this disk, all of the programs are contained in a single directory called *Kinematic M-Files*.

1.2.1 Running Programs on a Macintosh

To run the programs on a Macintosh, open MATLAB directly by double clicking on the MATLAB icon or by double clicking on one of the m-files in the *Kinematic M-Files* directory. The MATLAB application will then open. Next click on **File** on the main MATLAB menu, and move the mouse to **Open**. Under **Open**, locate the directory in which the programs are located (*Kinematic M-Files*) and double click on the file that you want to run in that directory. The file will then be opened in MATLAB.

In MATLAB, move the mouse to **Window** under the main menu bar and click on **MATLAB Command Window**. The MATLAB command window will then become the top window. At the MATLAB prompt (>>), type the name of the m-file you wish to run. Type only the name without the .m extension. The program will then open, and you can type in the input information described in the later chapters of this manual.

1.2.2 Running Programs Under Microsoft Windows on a PC

To run the programs on a PC, open MATLAB by moving the cursor to **Start** then to **Programs**, and then to **Matlab**. After MATLAB is running, set the path to the program directory moving the cursor to the **File** under the main MATLAB menu and click on **Set Path**. This brings up the Path Browser window. To find the folder containing the programs, click on **Browse**, and search the folders available. When the directory is found, identify it with the cursor, and click **Open**. In the Path Browwer, click on the directory which appears under Current Directory. In the main menu of the Path Browser, move the cursor to **File** and click on **Save Path** and then click on **Exit Path Browser**. This will return control to the main MATLAB window.

At the MATLAB prompt (>>), type the name of the m-file you wish to run. Type only the name without the .m extension. The program will then open, and you can type in the input information described in the later chapters of this manual.

1.2.3 MATLAB Graphics Window

As the programs are run, three windows will be of interest. The first is the MATLAB command window indicated above. The second is the program window which contains the program source code. This window needs to be open only when a program is changed. Each time a change is made in the program, it must be saved before the changes take affect.

The third window is the graphics window. The names of the windows that are open appear under **Window** in main MATLAB menu. If you expect to see graphics and none appears, the graphics window may be hidden under other open windows. To make the graphics window the current window, move the mouse to **Window** under the main MATLAB menu and locate Figure No.1. Clicking on Figure No. 1 will make the graphics window the top window.

1.2.4 Help in Using MATLAB

A brief overview of the use of MATLAB for kinematics problems is given in Section 14 of manual. Basic information on modifying the routines can be obtained from there; however, for detaileds, consult the MATLAB Users' Manual supplied by Mathworks. Alternatively, MATLAB has an excellent help facility. To obtain help on any topic in the library, simply type help and MATLAB will present a series of topics on which help may be obtained. By typing help and then the name of the topic, a description of that topic is displayed. Also, list of subtopics on which help can be obtained is displayed. If the name of the subtopic is known, it is possible to type help followed by the subtopic name anytime that the MATLAB prompt (>>) appears in the MATLAB window.

2.0 Programs for Chapter 2

2.A.1 MATLAB Routine for The Centrodes of a Four-Bar Mechanism

This appendix describes a MATLAB program for the generation of the moving and fixed centrodes for the coupler of a four-bar linkage. The procedure used in the program is to find the instant center I_{13} by locating the intersection C of the lines defined by the two frame mounted links (A*A and B*B in Fig. 2.A.1). The instant center I_{13} located at C is the location of two coincident points, C_1 fixed to the frame and C_3 fixed to the coupler. As the linkage moves, different locations are defined corresponding to different C's. The fixed centrode is defined by the plot of C_1 on the frame. The moving centrode is defined by the plot of C_3 on the coupler.

2.A.2 Line Intersection Routine (*lineintersect.m*)

The different locations of C_1 are defined by the coordinates at the intersection of A*A and B*B. In any given position, the coordinates (A^*_x, A^*_y) and (B^*_x, B^*_y) of A* and B*, respectively, will be defined by the fixed pivot locations. The coordinates (A_x, A_y) and (B_x, B_y) of A and B, respectively, will be defined by the position analysis of the linkage. The A*A line is given by

$$C_y = \left(\frac{A_y - A^*_y}{A_x - A^*_x}\right)(C_x - A_x) + A_y \quad \text{or}$$

$$(A_y - A^*_y)C_x - (A_x - A^*_x)C_y = (A_y - A^*_y)C_x - (A_x - A^*_x)A_y$$

Similarly, the B*B line can be represented by,

$$(B_y - B^*_y)C_x - (B_x - B^*_x)C_y = (B_y - B^*_y)C_x - (B_x - B^*_x)B_y$$

At the intersection, the coordinates (C_x, C_y) of C are given by solving

$$\begin{bmatrix}(A_y - A^*_y) & -(A_x - A^*_x)\\ (B_y - B^*_y) & -(B_x - B^*_x)\end{bmatrix}\begin{Bmatrix}C_x\\ C_y\end{Bmatrix} = \begin{Bmatrix}(A_y - A^*_y)C_x - (A_x - A^*_x)A_y\\ (B_y - B^*_y)C_x - (B_x - B^*_x)B_y\end{Bmatrix}$$

or

$$\begin{Bmatrix}C_x\\ C_y\end{Bmatrix} = \begin{bmatrix}(A_y - A^*_y) & -(A_x - A^*_x)\\ (B_y - B^*_y) & -(B_x - B^*_x)\end{bmatrix}^{-1}\begin{Bmatrix}(A_y - A^*_y)C_x - (A_x - A^*_x)A_y\\ (B_y - B^*_y)C_x - (B_x - B^*_x)B_y\end{Bmatrix} \qquad (2A.1)$$

The calculations necessary to solve Eq. (2A.1) are programmed using MATLAB in the routine *lineintersect.m*. The initial statement in the program is

```
function coords = lineintersect(a1,a2,b1,b2)
```

The input variables are:

```
a1 = a two component vector giving the x,y coordinates of the first point
     on first line.

a2 = a two component vector giving the x,y coordinates of the second point
      on first line.

b1 = a two component vector giving the x, y coordinates of the first point
     on second line.

b2 = a two component vector giving the x, y coordinates of the second point
```

```
on second line.
```

The returned vector is

coords = a three component vector. The first two components are the x, y coordinates of the intersection of the two lines. The third component is a flag. The flag is 0 if the lines are not parallel and 1 if the lines are parallel.

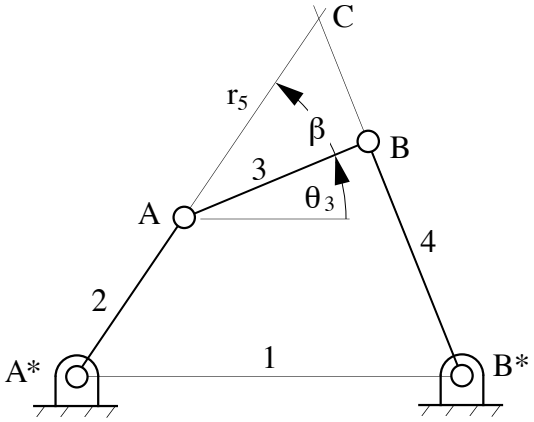


Fig. 2.A.1: Nomenclature for locating instant center at C

2.A.3 Locating C_3 Relative to Coupler

The location of C_3 relative to the coupler line AB is defined by the angle β and distance r_5. After the coordinates of C are defined in the position given, the angle β and distance r_5 are given by

$$\beta = \tan^{-1}\left(\frac{C_y - A_y}{C_x - A_x}\right) - \tan^{-1}\left(\frac{B_y - A_y}{B_x - A_x}\right) -$$

and

$$r_5 = \sqrt{(C_x - A_x)^2 + (C_y - A_y)^2}$$

The angle β and distance r_5 are defined for each position of the linkage. This defines the locates the centrode relative to the coupler line AB. In any given position, the moving centrode can be drawn relative to the frame by finding the coordinates of the different points of the moving centrode relative to the frame. If the subscript i identifies a point on the moving centrode, then the (x, y) coordinates of the point relative to the frame are

$$(x,y)_i = \left[(A_x + r_5\cos\theta_5)_i, (A_y + r_5\sin\theta_5)_i\right]$$

where

$$\theta_5 = \tan^{-1}\left(\frac{C_y - A_y}{C_x - A_x}\right)$$

2.A.4 Centrode Branches

When a crank-rocker mechanism is involved, the A*A and B*B will become parallel for two positions of the driving link. When this happens, the instant center at C will move to infinity. The centrodes are then infinitely large. If we locate a line alone A*A and define the plus direction at the A end and the minus

direction at the A* end, the centrode will go toward infinity in one direction and return from infinity in the other direction. The centrode will then be made up of two segments and it is necessary to ensure that the segments are properly aligned. This is done by the centrode drawing routine, *centrodes.m*

2.A.5 Centrode Drawing Routine (*centrodes.m*)

The MATLAB routine which actually draws the moving and fixed centrodes and animates the motion of the linkage is called *centrodes.m*. As the linkage moves, the moving centrode appear to roll on the fixed centrode. The variables which are input interactively to the program are

1) The number of cycles for which the animation is to be displayed

2) The length of the frame (A*B*) of the four-bar linkage

3) The length of the crank (A*A) of the four-bar linkage

4) The length of the coupler (AB) of the four-bar linkage

5) The length of the rocker (B*B) of the four-bar linkage

6) The angle between the frame and the x axis

In addition to *lineintersect.m*, the *centrodes.m* routine calls four other function routines (*fourbar.m*, *bushing.m*, *circle.m,* and *axisadjust.m*). The purposes of the routines are identified below, and more detailed descriptions are given in Chapter 3.

fourbar.m: This function analyzes a four-bar linkage for position, velocity, and acceleration.

bushing.m: This function returns the coordinates for drawing a bushing or frame pivot.

circle.m: This function returns the coordinates for a circle given the center location and radius

axisadjust.m: Function routine to adjust the axis limits so that the viewport and window in Matlab 5.0 is similar to that in Matlab 4.2 when the command "axis equal" is used

lineintersect.m: This function computes the coordinates of the intersection of two lines given two points on each line.

The program prints an input file which can be read in subsequent analyses to rerun the same case. The program initially asks the user is he/she is inputting the data through a file or through interactive input. A separate file name must be used for each analysis if the data file is to be used in subsequent analyses. If the data file specified already contains data, the new data are appended onto the end of the old. When the file is read as an input file, it is always read starting from the initial record in the file.

2.A.6 Sample run using *centrodes.m*

In the following, a copy of the input screen is given in Table 3.A.1 and the display in Fig. 3.A.2. The mechanism is a type 2 double rocker, and the centrode is made up of one part. Other examples are Figs. 2.63 and 2.64 where the centrodes are made up of multiple segments.

Table 2.A.1: Sample input and output for centrode analysis

```
        Centrode Analysis Program

Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (centrode.dat): temp.dat
Enter number of cycles (rev) [2]: 2
Enter the frame length [3]: 4
Enter the crank length [3]: 3
Enter the coupler length [2]: 2
Enter the rocker length [3]: 4.5
Enter the frame angle (deg) [0]: 0
I am working ...
repeat animation? y/n [y]: n
```

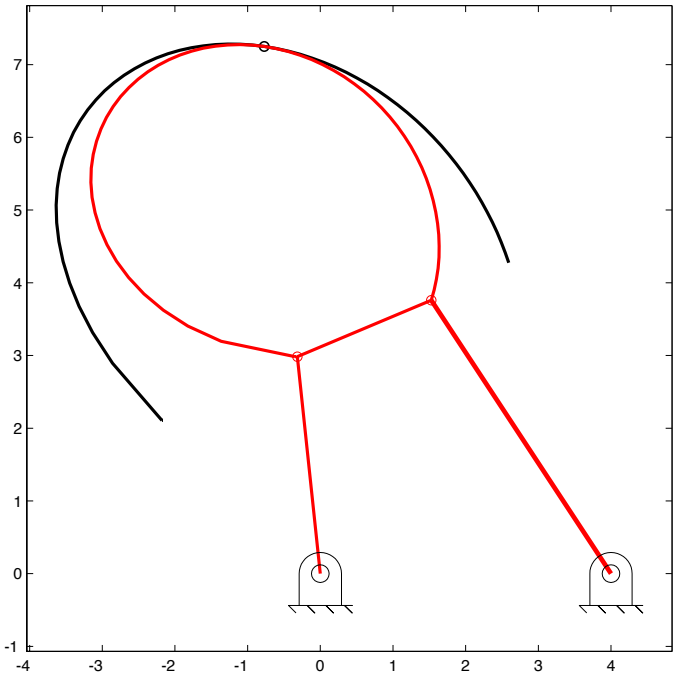


Fig. 2.A.2: Output from sample analysis corresponding to Table 2.A.1

3.0 Programs for Chapter 3: Mechanism Analysis

Chapter 3 includes 10 appendices which contain a description of the MATLAB function routines developed to illustrate the concepts in this Chapter.

3.A MATLAB Function For Four-Bar Linkage Analysis

The MATLAB function files are based on the nomenclature shown in Fig. 3.5 which is repeated here as Fig. 3.A.1 Three routines are given. The first is based on Table 3.1 which has the crank as the input. The routine is called *fourbar_cr.m*. The second is based on Table 3.1 when the coupler is the input. The routine is called *fourbar_co.m*. The first two routines analyze the mechanism in one position only. The third routine analyzes a four-bar linkage for a complete cycle of motion for the driver. The functions will be discussed separately.

3.A.1 Four-Bar Analysis Routine when the Crank Is the Driver (*fourbar_cr.m*)

The function *fourbar_cr.m* analyzes a four-bar mechanism when the crank is the driving link. The The initial statement in the function is

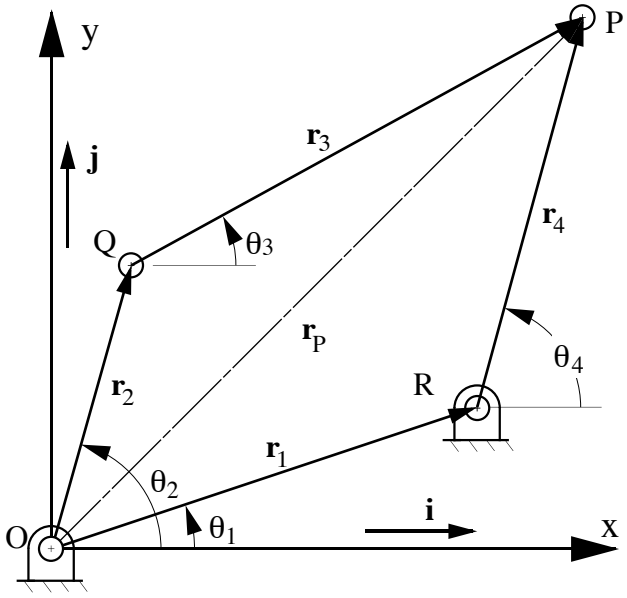


Fig. 3.A.1: Nomenclature for MATLAB Four-Bar Linkage Routine

```
function [values] = fourbar_cr(r1,r2,r3,r4,theta2,td2,tdd2, sigma, theta1,
                    flag)
```

The input variables are:

```
r1      = length of vector 1 (frame)
r2      = length of vector 2 (crank)
r3      = length of vector 3 (coupler)
r4      = length of vector 4 (slider offset)
theta2  = crank angle (deg.)
td2     = crank angular velocity (rad/sec)
tdd2    = crank angular acceleration (rad/sec^2)
sigma   = +1 or -1.  Identifies assembly mode
```

```
   theta1  = angle between line through fixed pivots and frame x axis (deg.)
   flag    = analysis flag.  If flag = 1, only a position analysis is
             conducted.  If flag = 2, both a position and velocity analysis is
             conducted.  If flag = 3, a position, velocity, and acceleration
             analysis is conducted.
```

The results are returned in the vector "values". The individual components of values are

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector angles (rad/sec)
values (13:16) = second derivatives of vector angles (rad/sec^2)
values (17:18) = x,y components of position of crank pin (point Q)
values (19:20) = x,y components of position of follower pin (point P)
values (21:22) = x,y components of velocity of crank pin (point Q)
values (23:24) = x,y components of velocity of follower pin (point P)
values (25:26) = x,y components of acceleration of crank pin (point Q)
values (27:28) = x,y components of acceleration of follower pin (point P)
values (29)    = assembly flag.  If values(29) = 0, mechanism cannot
                 assembled.
```

3.A.1.1 Source code for sample m-file (Examples 3.1 and 3.2) using *fourbar_cr.m*

A copy of the source code for a test program is given below. The results are given in Table 3.A.1.

```
% Solution to Examples 3.1 and 3.2

clear all;
r1=1;
r2=2;
r3=3.5;
r4=4;
theta1=0;
theta2=0;
td2=10;
tdd2=0;
sigma=1;
flag=3;

values = fourbar_cr(r1,r2,r3,r4,theta2,td2,tdd2,sigma,theta1,flag);
```

Table 3.A.1: Results for Examples 3.1 and 3.2

r	=	1.0000	2.0000	3.5000	4.0000
th	=	0	0	66.8676	53.5764
thd	=	0	10.0000	20.0000	20.0000
thdd	=	0	0	147.5798	85.4409
rq	=	2	0		
rp	=	3.3750	3.2186		
rdq	=	0	20		
rdp	=	-64.3720	47.5000		
rddq	=	-200	0		
rddp	=	-1225.0	-1084.5		

3.A.2 Four-Bar Analysis Routine when the Coupler Is the Driver (*fourbar_co.m*)

The function *fourbar_co.m* analyzes a four-bar mechanism when the coupler is the driving link. The initial statement in the function is

```
function [values] = fourbar_co(r1,r2,r3,r4,theta3,td3,tdd3, sigma, theta1,
                    flag)
```

The input variables are:

```
r1      = length of vector 1 (frame)
r2      = length of vector 2 (crank)
r3      = length of vector 3 (coupler)
r4      = length of vector 4 (slider offset)
theta3  = coupler angle (deg.)
td3     = coupler angular velocity (rad/sec)
tdd3    = coupler angular acceleration (rad/sec^2)
sigma   = +1 or -1.  Identifies assembly mode
theta1  = angle between line through fixed pivots and frame x axis (deg.)
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.  If flag = 2, both a position and velocity analysis is
          conducted.  If flag = 3, a position, velocity, and acceleration
          analysis is conducted.
```

The results are returned in the vector "values". The individual components of values are

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector angles (rad/sec)
values (13:16) = second derivatives of vector angles (rad/sec^2)
values (17:18) = x,y components of position of crank pin (point Q)
values (19:20) = x,y components of position of follower pin (point P)
values (21:22) = x,y components of velocity of crank pin (point Q)
values (23:24) = x,y components of velocity of follower pin (point P)
values (25:26) = x,y components of acceleration of crank pin (point Q)
values (27:28) = x,y components of acceleration of follower pin (point P)
values (29)    = assembly flag.  If values(29) = 0, mechanism cannot
                 assembled.
```

3.A.2.1 Source code for sample m-file (Examples 3.1 and 3.2) using *fourbar_co.m*

A copy of the source code for a test program is given below. The results are identical to those given in Table 3.A.1.

```
% Solution to inverse of Examples 3.1 and 3.2 using fourbar_co when the
% coupler is the input link.

clear all;
r1=1;
r2=2;
r3=3.5;
r4=4;
theta1=0;
theta3=66.867604;
td3=20.000000;
tdd3=147.579771;
sigma=-1;
flag=3;
```

```
values = fourbar_co(r1,r2,r3,r4,theta3,td3,tdd3,sigma,theta1,flag);
```

3.A.3 MATLAB Procedure for Analyzing a Four-Bar Mechanism for a Complete Cycle of Motion

3.A.3.1 Overview

The four-bar program, *fourbar_analysis.m*, analyzes a crank-rocker mechanism for position, velocity, and torques, and the graphical output draws a coupler curve. The nomenclature used in the program is shown in Fig. 3.A.2. The numerical analysis uses *fourbar_cr.m* and *fourbar_co.m*, and the graphical analysis uses *axisadjust.m*, *bushing.m*, *circle.m*, *fb_angle_limits_cr.m*, and *fb_angle_limits_co.m*. The utility routines *bushing.m* and *circle.m* draw bushings and circles, respectively. These routines are described in more detail in a later appendix. The routines *fb_angle_limits_cr.m* and *fb_angle_limits_co.m* determine the angle limits for the driver link. These are described in the following sections.

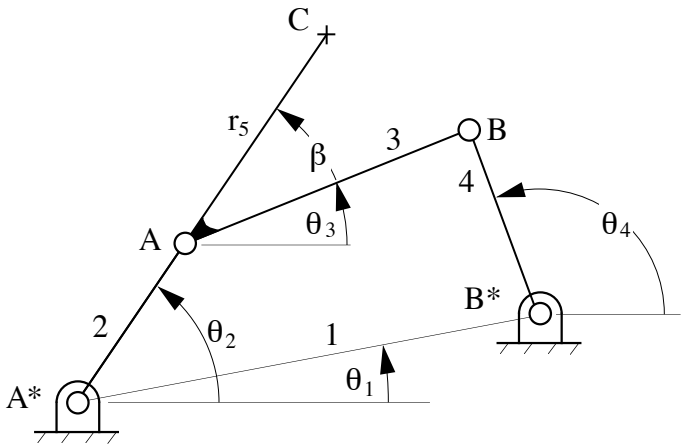


Fig. 3.A.2: Nomenclature for fourbar_analysis program

3.A.3.2 Angle Limits for Crank (r_2) as Input (*fb_angle_limits_cr.m*)

The angle limits when the crank is the driver depend on whether the linkage is a type I or type II linkage. Four conditions must be considered as shown in Fig. 3.A.3. These are discussed in the following.

3.A.3.2.1 Condition When $r_1 + r_2 \leq r_3 + r_4$ and $|r_1 - r_2| \geq |r_3 - r_4|$

This condition is indicated in Fig. 3.A.3a. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = 0$$

and

$$\theta_{max} = 2\pi$$

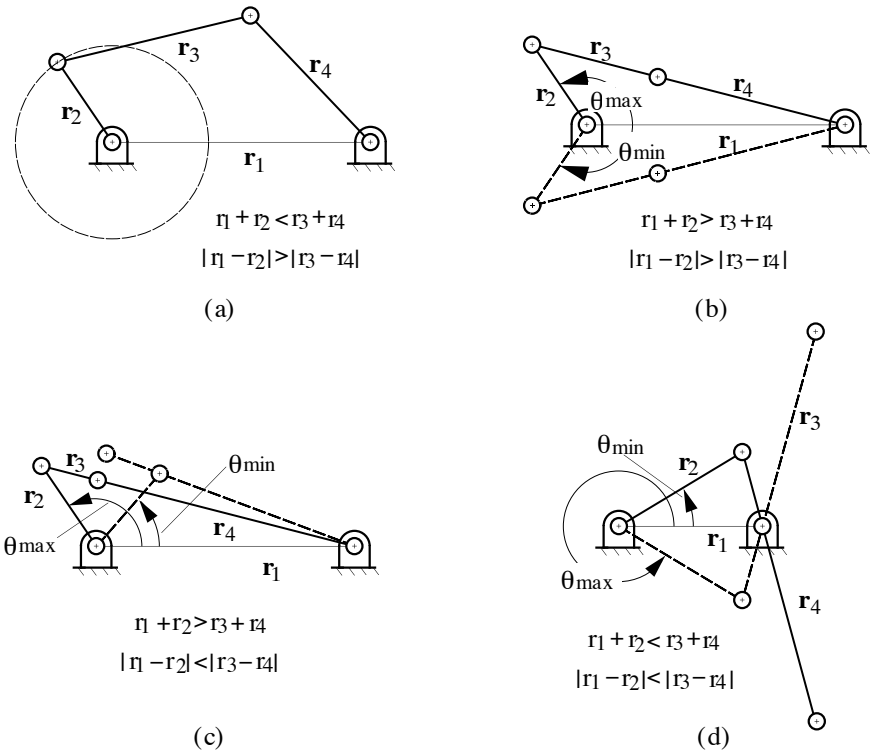


Fig. 3.A.3: Angle limits when the crank is the input link for a four-bar linkage

3.A.3.2.2 Condition When $r_1 + r_2 \geq r_3 + r_4$ and $|r_1 - r_2| \trianglelefteq r_3 - r_4|$

This condition is indicated in Fig. 3.A.3b. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = -\cos^{-1}\left[\{r_1^2 + r_2^2 - (r_4 + r_3)^2\} / (2r_1r_2)\right]$$

and

$$\theta_{max} = \cos^{-1}\left[\{r_1^2 + r_2^2 - (r_4 + r_3)^2\} / (2r_1r_2)\right]$$

3.A.3.2.3 Condition When $r_1 + r_2 \geq r_3 + r_4$ and $|r_1 - r_2| \trianglelefteq r_3 - r_4|$

This condition is indicated in Fig. 3.A.3c. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \cos^{-1}\left[\{r_1^2 + r_2^2 - (r_3 - r_4)^2\} / (2r_1r_2)\right]$$

and

$$\theta_{max} = \cos^{-1}\left[\{r_1^2 + r_2^2 - (r_4 + r_3)^2\} / (2r_1r_2)\right]$$

3.A.3.2.4 Condition When $r_1 + r_2 \leq r_3 + r_4$ and $|r_1 - r_2| \trianglelefteq r_3 - r_4|$

This condition is indicated in Fig. 3.A.3d. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \cos^{-1}\left[\{r_1^2 + r_2^2 - (r_3 - r_4)^2\} / (2r_1r_2)\right]$$

and

$$\theta_{max} = 2\pi - \cos^{-1}\left[\{r_1^2 + r_2^2 - (r_3 - r_4)^2\} / (2r_1r_2)\right]$$

3.A.3.3 Angle Limits for Coupler (r_3) as Input (*fb_angle_limits_co.m*)

The angle limits when the coupler is the driver again depend on whether the linkage is a type I or type II linkage. When the linkage is a type I (Grashof) linkage and the coupler (r_3) is the shortest link, the coupler can make a full rotation and the angle limits are defined by $0 \le \theta_3 \le 2\pi$. In other conditions, the coupler reaches an extreme position when r_2 and r_4 become parallel. Some of the conditions which must be considered are represented in Fig. 3.A.4 and are discussed in the following.

3.A.3.3.1 Condition When $r_1 + r_3 \le r_2 + r_4$ and $|r_1 - r_3| \le |r_2 - r_4|$

This condition is indicated in Fig. 3.A.4a. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = 0$$

and

$$\theta_{max} = 2\pi$$

3.A.3.3.2 Condition When $r_1 + r_3 \ge r_2 + r_4$ and $|r_1 - r_3| \le |r_2 - r_4|$

This condition is indicated in Fig. 3.A.4b. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = -\cos^{-1}\left[\{r_1^2 + r_3^2 - (r_4 + r_2)^2\} / (2r_1r_3)\right]$$

and

$$\theta_{max} = \cos^{-1}\left[\{r_1^2 + r_3^2 - (r_4 + r_2)^2\} / (2r_1r_3)\right]$$

3.A.3.3.3 Condition When $r_1 + r_3 \ge r_2 + r_4$ and $|r_1 - r_3| \le |r_2 - r_4|$

This condition is indicated in Fig. 3.A.4c. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \cos^{-1}\left[\{r_1^2 + r_3^2 - (r_2 - r_4)^2\} / (2r_1r_3)\right]$$

and

$$\theta_{max} = \cos^{-1}\left[\{r_1^2 + r_3^2 - (r_4 + r_2)^2\} / (2r_1r_3)\right]$$

3.A.3.3.4 Condition When $r_1 + r_3 \le r_2 + r_4$ and $|r_1 - r_3| \le |r_2 - r_4|$

This condition is indicated in Fig. 3.A.4d. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \cos^{-1}\left[\{r_1^2 + r_3^2 - (r_2 - r_4)^2\} / (2r_1r_3)\right]$$

and

$$\theta_{max} = 2\pi - \cos^{-1}\left[\{r_1^2 + r_3^2 - (r_2 - r_4)^2\} / (2r_1r_3)\right]$$

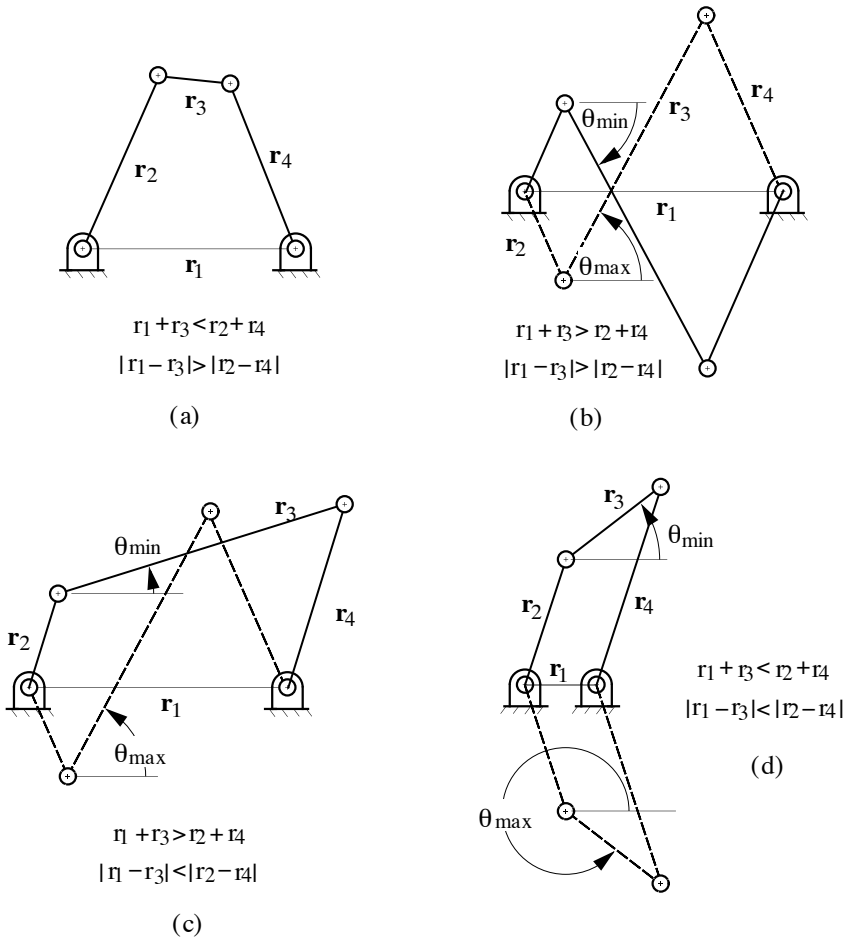


Fig. 3.A.4: Angle limits when the coupler is the input link for a four-bar linkage

3.A.3.4 Analysis Program(*fourbar_analysis.m*)

The inputs required by the program are defined in the following.

```
tt=number of cycles (rev)
r1= the frame length (A*B*)
Q1= the frame angle (deg)
r2=the crank length (AA*)
r3=the coupler length (AB)
r4= the rocker length (B*B)
cr1= distance from A to coupler point (AC or r5 in Fig. 3.A.2)
beta1= angle (β) from line AB to line AC (deg)
cpflag= 1 for coupler line and 2 for coupler triangle
driver=1 for crank driver and 2 for coupler driver
times= 1 for designated mode only and 2 for both modes
mode= the assembly mode (+1 or -1)
wdr= the angular velocity of driver link(rad/sec)
```

The results of a sample analysis are given in the following section.

3.A.3.5 Sample run using *fourbar_analysis.m*

In the following, a copy of the input screen is given in Table 3.A.2 and the plots are displayed in Figs. 3.A.5 and 3.A.6. In addition, a data file (fourbar.dat) is generated giving the results of the analysis at each position. Linear units are not given. Any units can be used as long as the set is consistent.

In the interactive mode, the programs prompts the user for each item of data. Default values have been included in the program, and these can be selected by simply pressing *return*. The default values are shown in square brackets []. The input data are printed to a data file that can be used in subsequent analyses in the "file-input" mode.

If a data file is available, the user needs only identify that a file input is to be used. The program then prompts for the name of the input file and reads the values for the input variables.

Table 3.A.2: Input and output corresponding to sample fourbar linkage analysis

```
        Four-Bar Linkage Analysis Program
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (fourbario.dat): manual.dat
Enter number of cycles (rev) [3]: 2
Enter the frame length [2]: 2
Enter the frame angle (deg) [0]: 10
Enter the crank length [2.5]: 1
Enter the coupler length [1.0]: 3
Enter the rocker length [2.5]: 3
Enter coupler point radius [0.5]: 1.5
Enter angle from coupler line to coupler point (deg) [0]: -45

    Flag to determine how coupler is to be drawn
Enter 1 for coupler line and 2 for coupler triangle [1]: 1

    Flag to determine driver link (crank or coupler)
Enter 1 for crank driver and 2 for coupler driver [1]: 1

    Flag to determine if both linkage modes are to be computed
Enter 1 for designated mode only and 2 for both modes [1]: 2
Enter the assembly mode (+1 or -1) [-1]: -1
Enter the angular velocity of driver link(rad/sec) [5]: 20
I am working ...
Repeat animation? y/n [y]: y
Change animation speed? (+/-/0) [-]:-
Repeat animation? y/n [y]: n
```

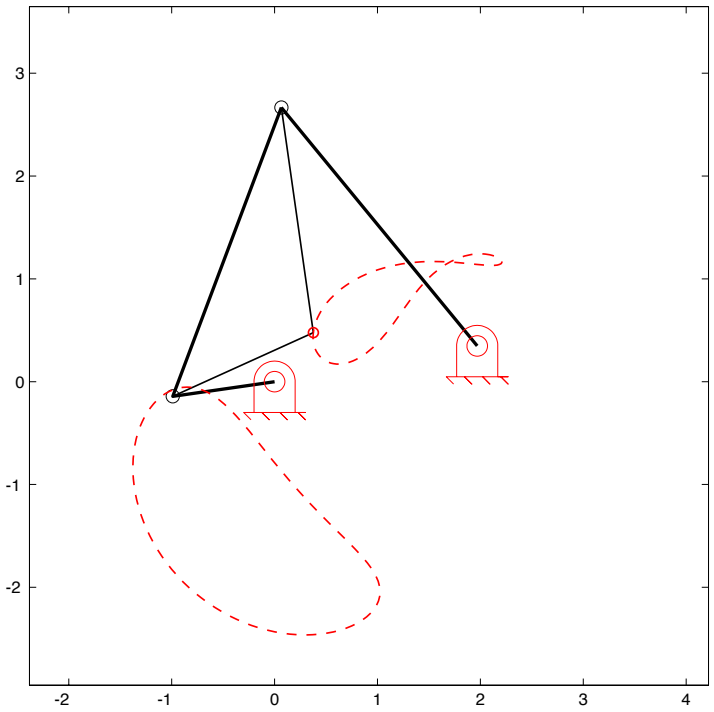


Fig. 3.A.5: Plot of fourbar linkage and coupler curves for both assembly modes.

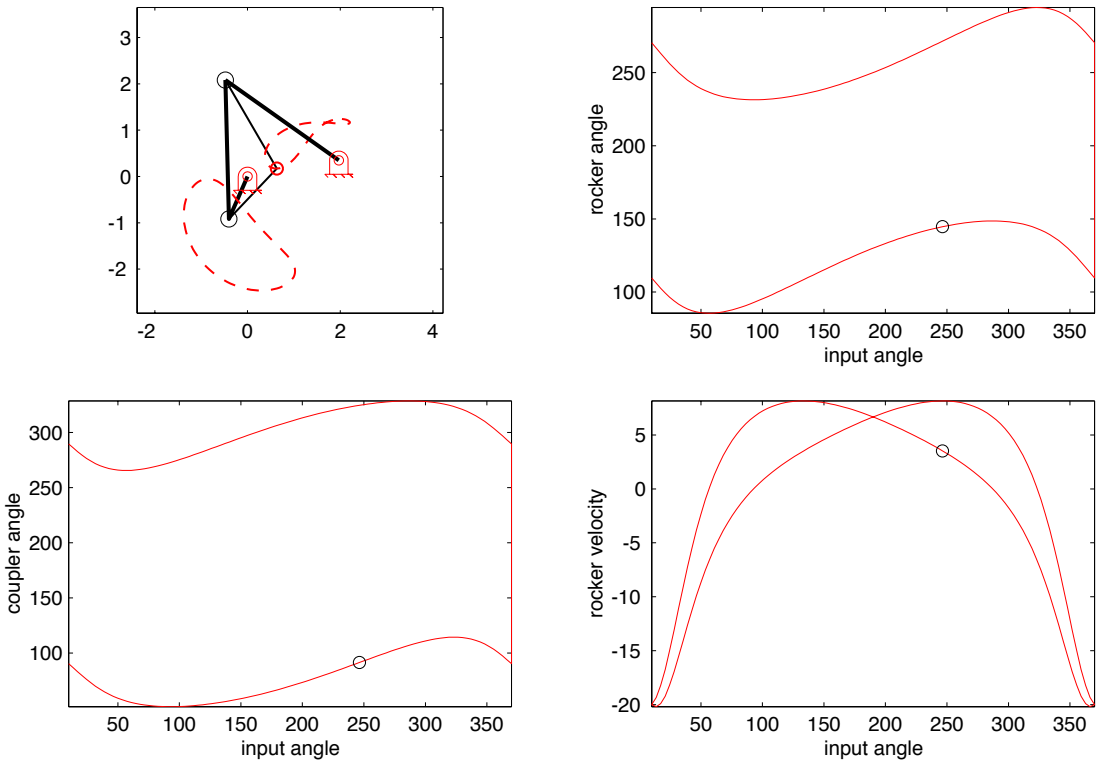


Fig. 3.A.6: Plots of position, velocity, and torque for both assembly modes

3.B MATLAB Function For Rigid Body Analysis

The MATLAB function files are based on the nomenclature shown in Fig. 3.9 which is repeated here as Fig. 3.B.1 Two routines are given. The first is based on the assumption that the kinematic properties of two points are known. The second is based on the assumption that the angular velocity and acceleration are known. The first routine is called *rbody1.m* and the second is called *rbody2.m*. The two functions will be discussed separately.

3.B.1 Rigid Body Routine When A and B Are Known (*rbody1.m*)

The function *rbody1.m* determines the kinematic properties of a point (C) on a rigid body when the kinematic properties of two points (A and B) are already known.

The initial statement in the function is:

```
function [values] = rbody1(r6, beta, rax, ray, rbx, rby, vax, vay, vbx,
                    vby, aax, aay, abx, aby, flag)
```

The input variables are:

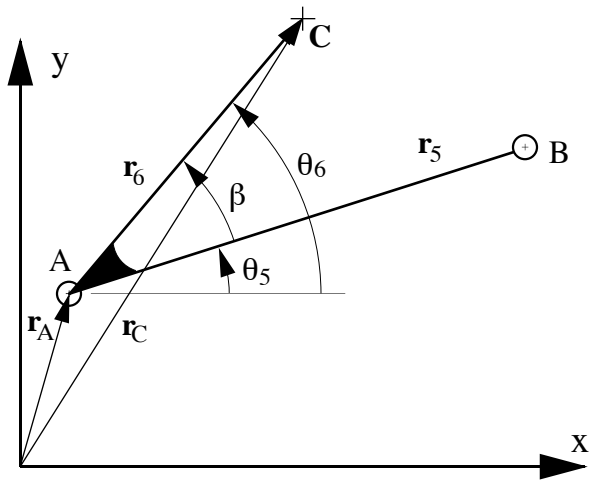


Fig. 3.B.1: Nomenclature for MATLAB Rigid-Body Routine

```
r6      = length of vector from point A to point C
beta    = angle from line through AB to the line AC (degrees)
rax     = x coordinate of point A
ray     = y coordinate of point A
rbx     = x coordinate of point B
rby     = y coordinate of point B
vax     = x component of velocity of A
vay     = y component of velocity of A
vbx     = x component of velocity of B
vby     = y component of velocity of B
aax     = x component of acceleration of A
aay     = y component of acceleration of A
abx     = x component of acceleration of B
aby     = y component of acceleration of B
flag    = analysis flag.  If flag = 1, only a position analysis is conducted.
          If flag = 2, both a position and velocity analysis is conducted.
          If flag = 3, a position, velocity, and acceleration analysis is
          conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:2) = x,y components of point C
values (3:4) = x, y components of velocity of point C
values (5:6) = x, y components of acceleration of point C
values (7)   = angle between fixed x axis and line AC(degrees)
values (8)   = angular velocity of rigid body (rad/sec)
values (9)   = angular acceleration of rigid body (rad/sec^2)
```

3.B.1.1 Source Code for Sample m-file (Examples 3.3) Using *rbody1.m*

A copy of the source code for a test program is given below. The results are given in Table 3.B.1.

```
% Solution to Example 3.3 using rbody1.m
clear all;
r6=2.236;
theta=66.87;
beta=26.565;
rax=2;
ray=0;
vax=0;
vay=20;
aax=-200;
aay=0;
rbx=3.375;
rby=3.2186;
vbx=-64.372;
vby=47.5;
abx=-1225.0;
aby=-1084.5;
flag=3;
values = rbody1(r6,beta,rax,ray,rbx,rby,vax,vay,vbx,vby,aax,aay,abx,aby,flag);
```

Table 3.B.1: Results for Example 3.3 using *rbody1.m*

```
rc =       1.8661    2.2320
vc =     -44.6398   17.3224
ac =    -475.8443 -912.5533
theta6 =  93.4326
omega =        20
alpha =  147.5797
```

3.B.2 Rigid Body Routine When A and B Are Known (*rbody2.m*)

The function *rbody2.m* determines the kinematic properties of a point (C) on a rigid body when the position of one point (A) and the angular position, velocity, and acceleration of the link are already known. The initial statement in the function is:

```
function [values] = rbody2(r6,beta,rax,ray,vax,vay,aax,aay,theta,omega,
                    alpha,flag)
```

The input variables are:

```
r6      = length of vector from point A to point C
beta    = angle from line through AB to the line AC (degrees)
rax     = x coordinate of point A
ray     = y coordinate of point A
vax     = x component of velocity of A
vay     = y component of velocity of A
aax     = x component of acceleration of A
aay     = y component of acceleration of A
theta   = angle from horizontal to axix from which beta is measured
          (degrees)
omega   = angular velocity of rigid body (rad/sec)
alpha   = angular acceleration of rigid body (rad/sec^2)
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.  If flag = 2, both a position and velocity analysis is
          conducted.  If flag = 3, a position, velocity, and acceleration
          analysis is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:2)   = x,y components of point C
values (3:4)   = x,y components of velocity of point C
values (5:6)   = x,y components of acceleration of point C
values (7)     = angle between fixed x axis and line AC(degrees)
```

3.B.2.1 Source Code for Sample m-file (Example 3.3) Using *rbody2.m*

A copy of the source code for a test program is given below. The results are given in Table 3.B.2.

```
% Solution to Example 3.3 using rbody2.m

clear all;
r6=2.236;
theta=66.87;
beta=26.565;
omega=20;
alpha=147.56;
rax=2;
ray=0;
vax=0;
vay=20;
aax=-200;
aay=0;
flag=3;
values = rbody2(r6,beta,rax,ray,vax,vay,aax,aay,theta,omega,alpha,flag);
```

Table 3.B.2: Results for Example 3.3 using *rbody2.m*

```
rc     =        1.8660    2.2320
vc     =      -44.6397   17.3205
ac     =     -475.7624 -912.5621
theta6 =       93.4350
```

3.C MATLAB Functions For Slider-Crank Analysis

The MATLAB function files are based on the nomenclature shown in Fig. 3.14 which is repeated here as

Fig. 3.C.1 Four routines are given. The first is based on Table 3.4 with the crank as the input. The second is based on Table 3.4 with the coupler as the input, and the third is based on Table 3.5 which has the slider as the input. The first routine is called *sldcrks.m*, the second is called *sldcrkco.m* and the third is called *sldcrkc.m*. The fourth routine analyzes the slider-crank mechanism for a full range of motion. That routine is called *slidercrank_anal.m*. The functions will be discussed separately.

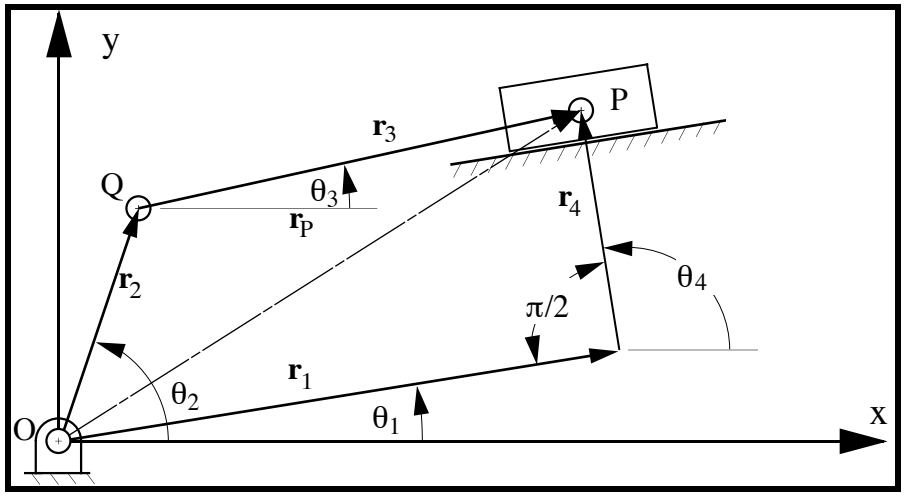


Fig. 3.C.1: Nomenclature for MATLAB Function Routines.

3.C.1 Slider-Crank Routine when the Crank Is Driver (*sldcrkc.m*)

The function *sldcrkc.m* analyzes a slider crank mechanism when the crank is the driving link. The initial statement in the function is:

```
function [values] = sldcrkc(r2,r3,r4,theta2,td2,tdd2,sigma,theta1,flag)
```

The input values are:

```
r2     = length of vector 2 (crank)
r3     = length of vector 3 (coupler)
r4     = length of vector 4 (slider offset)
theta2 = crank angle (degrees)
td2    = crank angular velocity (rad/sec)
tdd2   = crank angular acceleration (rad/sec^2)
sigma  = +1 or -1.  Identifies assembly mode
theta1 = angle between slider velocity and frame x axis (degrees).
flag   = analysis flag.  If flag = 1, only a position analysis is
         conducted.  If flag = 2, both a position and velocity analysis is
         conducted.  If flag = 3, a position, velocity, and acceleration
         analysis is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of crank pin (point Q)
```

```
values (27:28) = x,y components of position of piston pin (point P)
values (29:30) = x,y components of velocity of crank pin (point Q)
values (31:32) = x,y components of velocity of piston pin (point P)
values (33:34) = x,y components of acceleration of crank pin (point Q)
values (35:36) = x,y components of acceleration of piston pin (point P)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

3.C.1.1 Source Code for Sample m-file (Example 3.4) Using *sldcrkc.m*

A copy of the source code for a test program is given below. The results are given in Table 3.C.1.

```
% Solution to Example 3.4

clear all;
r2=5;
r3=8;
r4=0;
theta1=0;
theta2=45;
fact=pi/180;
td2=10;
tdd2=0;
sigma=1;
flag=3;

values = sldcrkc(r2,r3,r4,theta2,td2,tdd2,sigma,theta1,flag);
```

Table 3.C.1: Results for Example 3.4 using *sldcrkc.m*

```
r    =   10.7119    5.0000    8.0000         0
th   =         0   45.0000  -26.2278   90.0000
rd   =   52.7737         0         0         0
thd  =         0   10.0000   -4.9266         0
rdd  =  395.8309         0         0         0
thdd =         0         0   37.3086         0
rq   =    3.5355    3.5355
rp   =   10.7119    0.0000
rdq  =  -35.3553   35.3553
rdp  =  -52.7737         0
rddq = -353.5534 -353.5534
rddp = -395.8309    0.0000
```

3.C.2 Slider-Crank Routine when the Coupler Is the Driver (*sldcrkco.m*)

The function *sldcrkco.m* analyzes a slider crank mechanism when the coupler is the driving link. The initial statement in the function is:

```
function [values] = sldcrkco(r2,r3,r4,theta3,td3,tdd3,sigma,theta1,flag)
```

The input values are:

```
r2     = length of vector 2 (crank)
r3     = length of vector 3 (coupler)
r4     = length of vector 4 (slider offset)
theta3 = coupler angle (degrees)
td3    = coupler angular velocity (rad/sec)
tdd3   = coupler angular acceleration (rad/sec^2)
sigma  = +1 or -1.  Identifies assembly mode
theta1 = angle between slider velocity and frame x axis (degrees).
flag   = analysis flag.  If flag = 1, only a position analysis is
         conducted.  If flag = 2, both a position and velocity analysis is
         conducted.  If flag = 3, a position, velocity, and acceleration
         analysis is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of crank pin (point Q)
values (27:28) = x,y components of position of piston pin (point P)
values (29:30) = x,y components of velocity of crank pin (point Q)
values (31:32) = x,y components of velocity of piston pin (point P)
values (33:34) = x,y components of acceleration of crank pin (point Q)
values (35:36) = x,y components of acceleration of piston pin (point P)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

3.C.2.1 Source Code for Sample m-file (Example 3.4) Using *sldcrkco.m*

A copy of the source code for a test program is given below. The results are identical to those given in Table 3.C.1.

```
% Solution to inverse of Example 3.4 when coupler is the driver
```

% Solution to inverse of Example 3.4 when coupler is the input

```
clear;
r2=5;
r3=8;
r4=0;
theta1=0;
theta3=-26.227837;
fact=pi/180;
td3=-4.926646;
tdd3=37.308584;
sigma=1;
flag=3;

values = sldcrkco(r2,r3,r4,theta3,td3,tdd3,sigma,theta1,flag);
```

3.C.3 Slider-Crank Routine when the Slider Is the Driver (*sldcrks.m*)

The function *sldcrkc.m* analyzes a slider crank mechanism when the slider is the driving link. The initial

statement in the function is:

```
function [values] = sldcrks(r1,r2,r3,r4,rd1,rdd1,sigma,theta1,flag)
```

The input values are:

```
r1     = length of vector 1
r2     = length of vector 2 (crank)
r3     = length of vector 3 (coupler)
r4     = length of vector 4 (slider offset)
rd1    = slider velocity
rdd1   = slider acceleration
sigma  = +1 or -1.  Identifies assembly mode
theta1 = angle between slider velocity and frame x axis.
flag   = analysis flag.  If flag = 1, only a position analysis is
         conducted.  If flag = 2, both a position and velocity analysis is
         conducted.  If flag = 3, a position, velocity, and acceleration
         analysis is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of crank pin (point Q)
values (27:28) = x,y components of position of piston pin (point P)
values (29:30) = x,y components of velocity of crank pin (point Q)
values (31:32) = x,y components of velocity of piston pin (point P)
values (33:34) = x,y components of acceleration of crank pin (point Q)
values (35:36) = x,y components of acceleration of piston pin (point P)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

3.C.3.1 Source Code for Sample m-file (Examples 3.5) Using *sldcrks.m*

A copy of the source code for a test program is given below. The results are given in Table 3.C.2.

```
% Solution to Example 3.5

clear;
r1=10.75;
r2=5;
r3=8;
r4=0;
theta1=0;
rd1=50;
rdd1=400;
sigma=1;
flag=3;

values = sldcrks(r1,r2,r3,r4,rd1,rdd1,sigma,theta1,flag);
```

Table 3.C.2: Results for Example 3.5 using *sldcrks.m*

```
r    =   10.7500    5.0000    8.0000         0
th   =         0   44.5850  -26.0228   90.0000
rd   =        50         0         0         0
thd  =         0    9.5267   -4.7190         0
rdd  =       400         0         0         0
thdd =         0    6.4930   30.2213         0
rq   =    3.5610    3.5098
rp   =   10.7500         0
rdq  =  -33.4370   33.9249
rdp  =       -50         0
rddq = -345.9796 -295.4210
rddp = -400.0000         0
```

3.C.4 MATLAB Procedure for Analyzing Slider-Crank Mechanism for a Complete Cycle (*slidercrank_anal.m*)

3.C.4.1 Overview

In the slider-crank program, *slidercrank_anal.m*, analyzes a crank-rocker mechanism for position and velocity for a complete cycle and draws a coupler curve. The nomenclature used in the program is shown in Fig. 3.C.2. The numerical analysis uses *sldcrkc.m*, *sldcrks.m*, *sldcrkco.m*, *sc_angle_limits_cr.m, sc_angle_limits_co.m*, *sc_angle_limits_sl.m*, and the graphical analysis uses *axisadjust.m*, *bushing.m*, *frameline.m*, and *circle.m*. The routines *sc_angle_limits_cr.m, sc_angle_limits_co.m*, and *sc_angle_limits_sl.m* determine the limits for the input variables. These are described in the following sections.

3.C.4.2 Angle Limits for Crank (r_2) as Input (*sc_angle_limits_cr.m*)

The angle limits when the crank is the driver depend on whether the crank can make a full rotation. If $r_2 < r_3 - r_4$, a full rotation is possible and the angle limits are give by $0 \le \theta_2 \le 2\pi$. If $r_2 > r_3 - r_4$, the limiting positions of the crank will be those resulting when the coupler is perpendicular to the slider line. This is shown in Fig. 3.C.3. As indicated, four different cases must be considered.

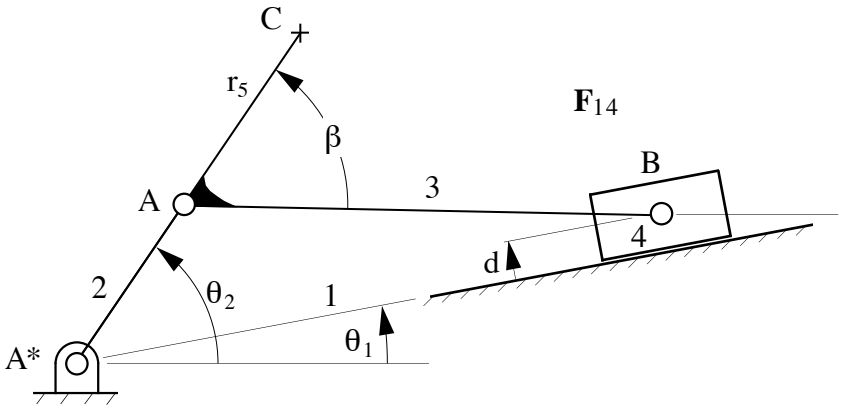


Fig. 3.C.2: Nomenclature for slider-crank analysis program

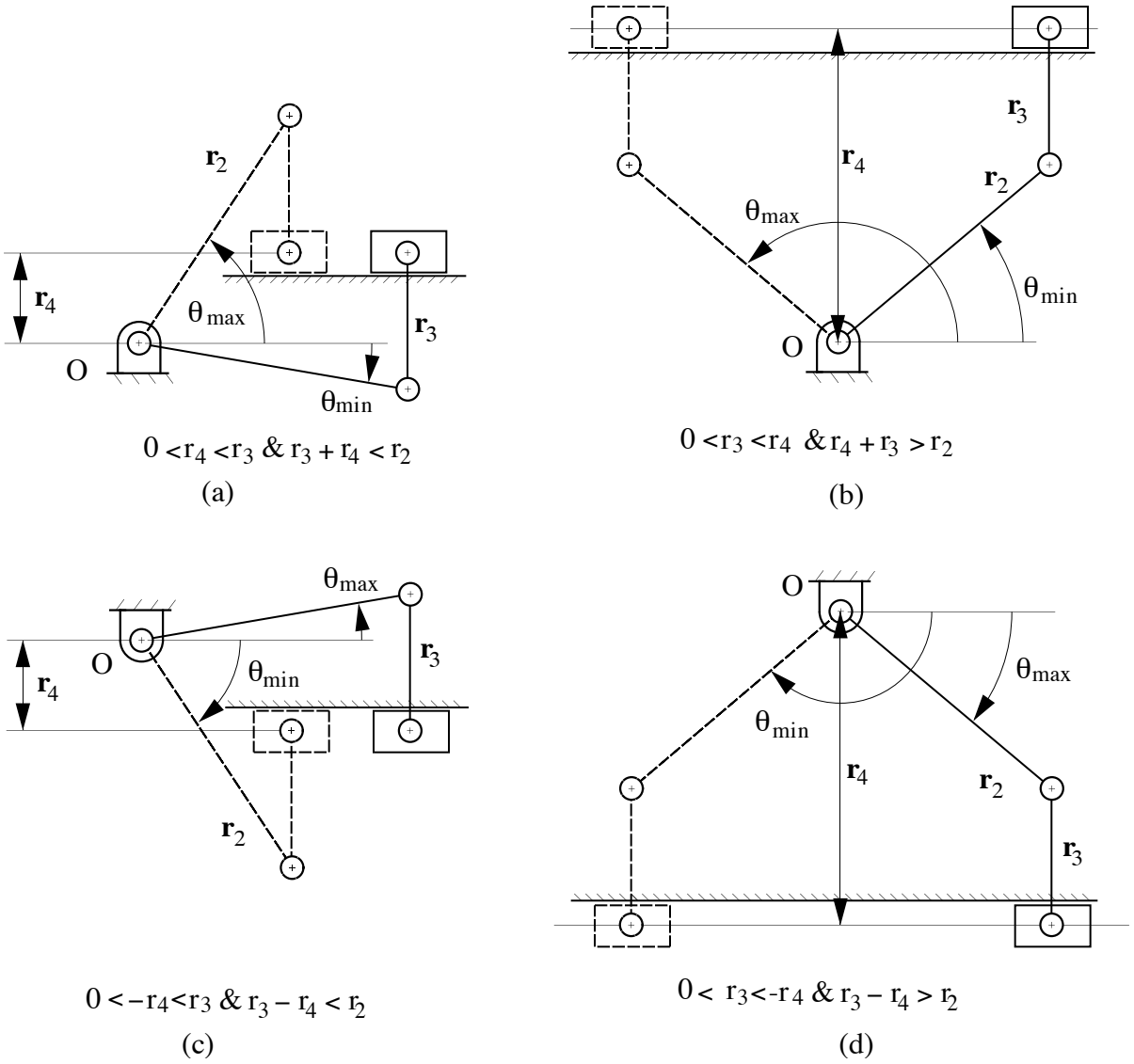


Fig. 3.C.3: Limiting positions when the crank is the driver

3.C.4.2.1 Mechanism When $0 \le r_4 \le r_3$ and $r_3 + r_4 \le r_2$

This condition is indicated in Fig. 3.A.3a. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \theta_1 + \sin^{-1}[(r_4 - r_3) / r_2]$$

and

$$\theta_{max} = \theta_1 + \sin^{-1}[(r_4 + r_3) / r_2]$$

3.C.4.2.2 Mechanism When $0 \le r_3 \le r_4$ and $r_3 + r_4 \ge r_2$

This condition is indicated in Fig. 3.A.3b. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \theta_1 + \sin^{-1}[(r_4 - r_3) / r_2]$$

and

$$\theta_{max} = \theta_1 + \pi - \sin^{-1}[(r_4 - r_3) / r_2]$$

3.C.4.2.3 Mechanism When $0 \le -r_4 \le r_3$ and $r_3 - r_4 \le r_2$

This condition is indicated in Fig. 3.A.3b. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \theta_1 + \sin^{-1}[(r_4 - r_3) / r_2]$$

and

$$\theta_{max} = \theta_1 + \sin^{-1}[(r_3 + r_4) / r_2]$$

3.C.4.2.4 Mechanism When $0 \le r_3 \le -r_4$ and $r_3 - r_4 \ge r_2$

This condition is indicated in Fig. 3.A.3b. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \theta_1 - \pi - \sin^{-1}[(r_3 + r_4) / r_2]$$

and

$$\theta_{max} = \theta_1 + \sin^{-1}[(r_3 + r_4) / r_2]$$

3.C.4.3 Angle Limits for the Coupler (r_3) as Input (*sc_angle_limits_co.m*)

The angle limits when the coupler is the driver are represented in Fig. 3.C.4. If $r_2 \ge r_3 + |r_4|$, the coupler link can rotate for a full revolution. Therefore, the angle limits are given by $0 \le \theta_3 \le 2\pi$. When $r_2 < r_3 + |r_4|$, the limiting condition occurs when the crank (link 2) is perpendicular to the slider line. This is shown in Fig. 3.C.4. Four different cases must be considered, and these are indicated in the following.

3.C.4.3.1 Mechanism When $0 \le r_4$ and $r_2 + r_4 \ge r_3$

This condition is indicated in Fig. 3.C.4a. The angle limits are given by θ_{min} and θ_{max} where

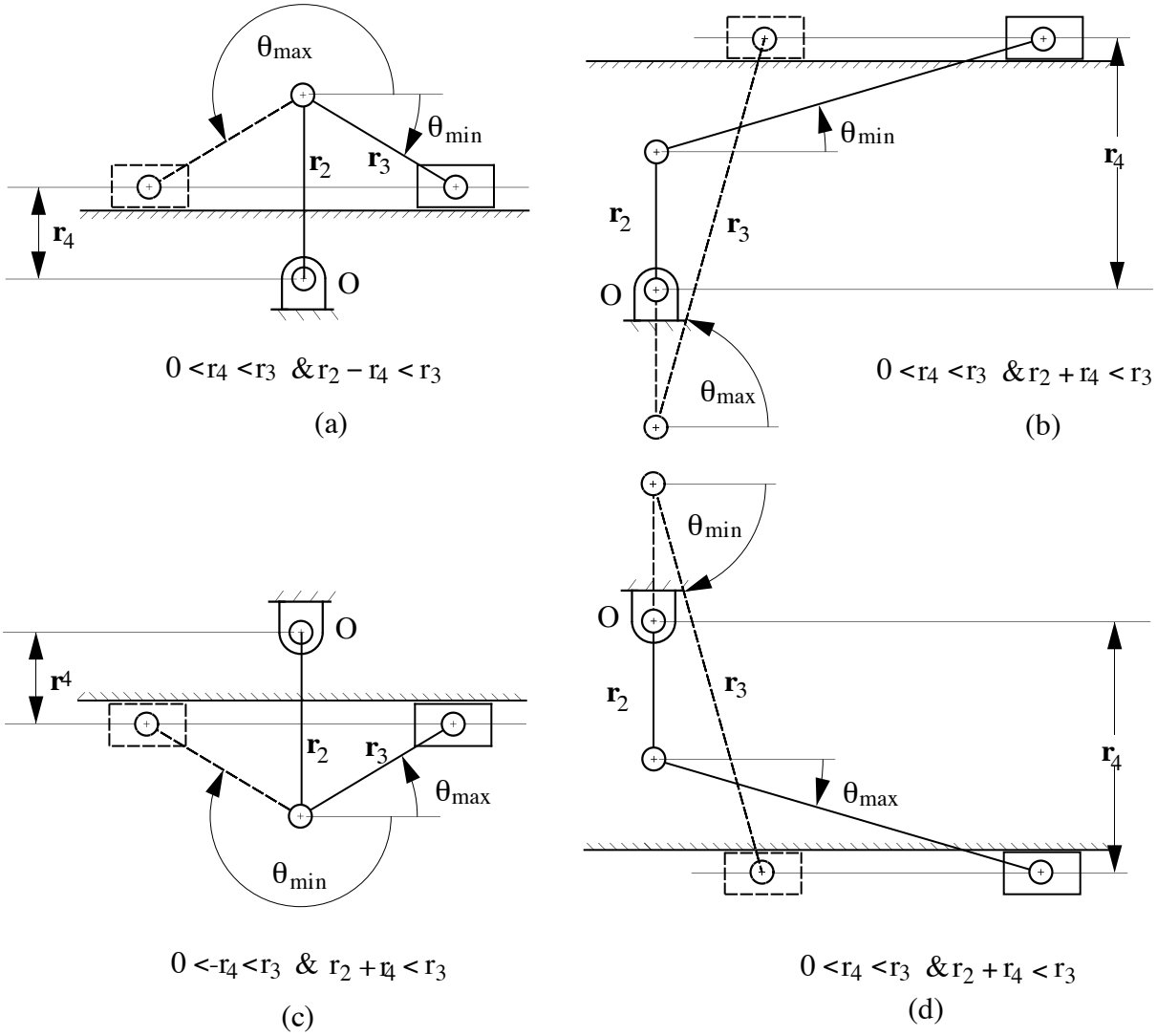


Fig. 3.C.4: Limiting positions when the coupler is the driver

$$\theta_{min} = \theta_1 + \sin^{-1}[(r_4 - r_2)/r_3]$$

and

$$\theta_{max} = \theta_1 + \pi - \sin^{-1}[(r_4 - r_2)/r_3]$$

3.C.4.3.2 Mechanism When $0 \le r_4$ and $r_2 + r_4 \le r_3$

This condition is indicated in Fig. 3.C.3b. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \theta_1 + \sin^{-1}[(r_4 - r_2)/r_3]$$

and

$$\theta_{max} = \theta_1 + \sin^{-1}[(r_4 + r_2)/r_3]$$

3.C.4.3.3 Mechanism When $0 \geq r_4$ and $r_2 + r_4 < r_3$

This condition is indicated in Fig. 3.C.3c. The angle limits are given by θ_{min} and θ_{max} where

$$\theta_{min} = \theta_1 - \pi - \sin^{-1}[(r_4 + r_2)/r_3]$$

and

$$\theta_{max} = \theta_1 + \sin^{-1}[(r_2 + r_4)/r_3]$$

3.C.4.3.4 Mechanism When $0 \geq r_4$ and $r_2 - r_4 \leq r_3$

This condition is indicated in Fig. 3.C.3d. The angle limits are given by θ_{min} and θ_{max} where

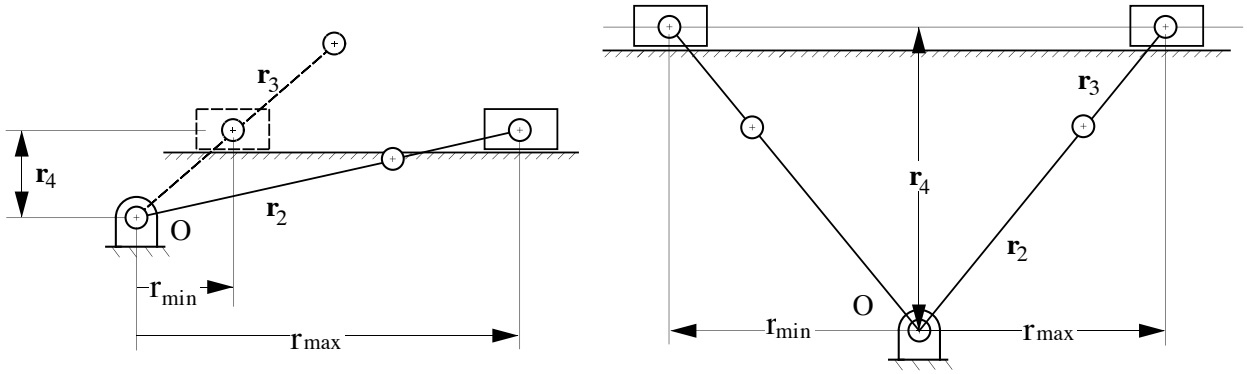
$$\theta_{min} = \theta_1 + \sin^{-1}[(r_4 - r_2)/r_3]$$

and

$$\theta_{max} = \theta_1 + \sin^{-1}[(r_2 + r_4)/r_3]$$

3.C.4.4 Limits when the Slider (r_1) as Input (*sc_angle_limits_sl.m*)

The limits for r_1 occur when links 2 and 3 are colinear. The two cases which must be considered are shown in Fig. 3.C.5. The two cases will be discussed separately. Note that the results given apply for both positive and negative values of r_4.



$|r_4 - r_3| > r_2$ (a)

$|r_4 - r_3| < r_2$ (b)

Fig. 3.C.5: Limiting positions when the slider is the driver

3.C.4.4.1 Mechanism When $|r_2 - r_3| > r_4$

This condition is indicated in Fig. 3.C.5a. The limits for r_1 are given by r_{min} and r_{max} where

$$r_{min} = \sqrt{(r_2 - r_3)^2 - r_4^2}$$

and

$$r_{max} = \sqrt{(r_2 + r_3)^2 - r_4^2}$$

3.C.4.4.2 Mechanism When $|r_2 - r_3| < r_4$

This condition is indicated in Fig. 3.C.5b. The limits for r_1 are given by r_{min} and r_{max} where

$$r_{min} = -\sqrt{(r_2 + r_3)^2 - r_4^2}$$

and

$$r_{max} = \sqrt{(r_2 + r_3)^2 - r_4^2}$$

3.C.4.5 Slider-Crank Analysis Routine *slidercrank_anal.m*

The inputs required by the program are defined in the following.

```
tt      = number of cycles for which animation is to be shown
d       = slider offset
θ1      = slider line angle measured CCW from the horizontal x axis (deg)
r2      = length of crank (A*A)
r3      = length of coupler (AB)
cr1     = distance from A to coupler point (AC or r5 in Fig. 3.C.1)
beta    = angle (β) from line AB to line AC (deg)
driver  = flag designating link which is to be the input.  (driver = 1 for
            for crank, driver = 2 for coupler, and driver = 3 for slider)
mode    = assembly mode (+1 or -1)
w2      = angular velocity of crank (when driver = 1)
w3      = angular velocity of coupler (when driver = 2)
r1d     = linear velocity of slider (when driver = 3)
```

The results of a sample analysis are given in the following section.

3.C.4.6 Sample run using *slidercrank_anal.m*

In the following, a copy of the input screen is given in Table 3.C.3 and the plots are displayed in Figs. 3.C.6 and 3.C.7. In addition, a data file (slidercrank.dat) is generated giving the results of the analysis at each position. Linear units are not given. Any units can be used as the set is consistent.

In the interactive mode, the programs prompts the user for each item of data. Default values have been included in the program, and these can be selected by simply pressing *return*. The default values are shown in square brackets []. The input data are printed to a data file that can be used in subsequent analyses in the "file-input" mode.

If a data file is available, the user needs only identify that a file input is to be used. The program then prompts for the name of the input file and reads the values for the input variables.

Table 3.C.3: Input and output corresponding to sample analysis

```
                Slider-Crank Analysis Program
      Enter 1 for file input and 2 for interactive input [1]: 2
      Enter input file name (slidercrkio.dat): manual.dat
      Enter number of cycles (rev) [3]: 2
      Enter slider-line offset [1]: -0.2
      Enter the slider line angle (deg) [30]: -30
      Enter the crank length [3]: 1
      Enter the coupler length [8]: 3
      Enter coupler point radius [6]: 2
      Enter angle from coupler line to coupler point (deg) [20]: 45
      Enter 1 for crank input, 2 for coupler input, and 3 for slider input [2]: 1
      Enter the assembly mode (+1 or -1) [1]: 1
          Flag to determine if both linkage modes are to be computed
      Enter 1 for designated mode only and 2 for both modes [2]: 1
      Enter the angular velocity of crank(rad/sec) [5]: 100
      I am working ...
      Repeat animation? y/n [y]: n
»
```

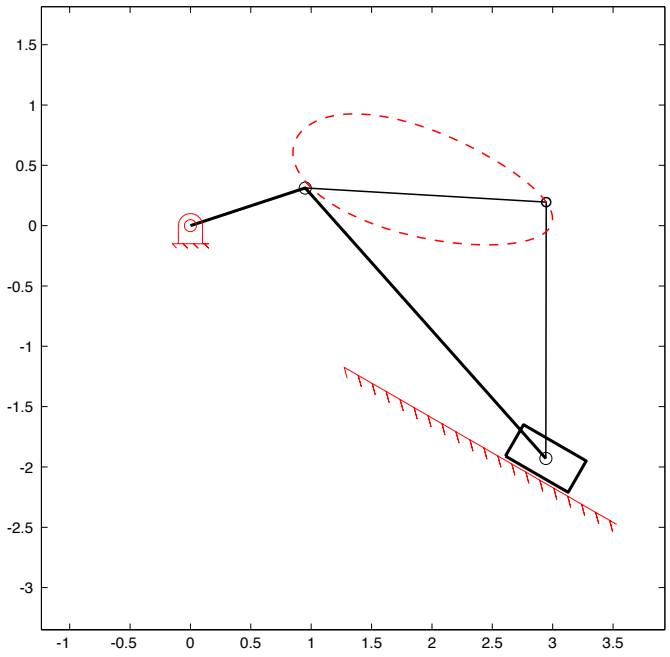


Fig. 3.C.6: Plot of slider-crank mechanism and coupler curve

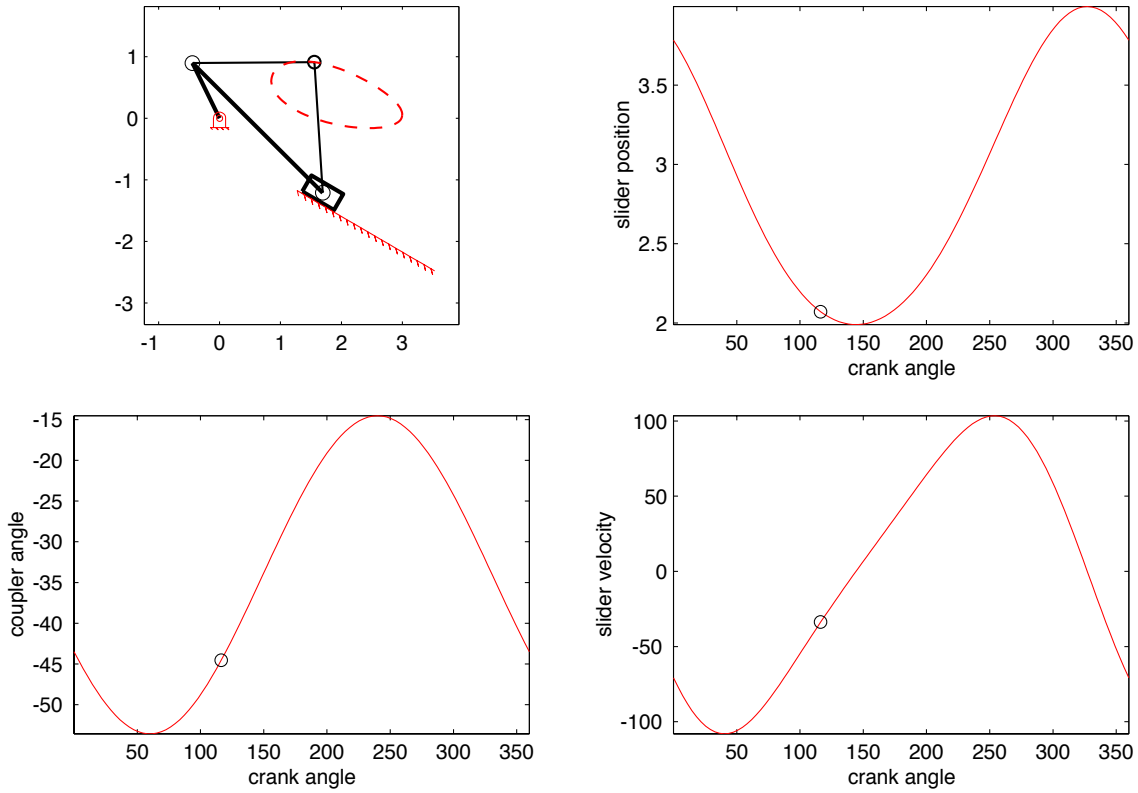


Fig. 3.C.7: Kinematic analysis of linkage

3.D MATLAB Functions For Inverted Slider-Crank Analysis

The MATLAB function files are based on the nomenclature shown in Fig. 3.23 which is repeated here as Fig. 3.D.1 Three routines are given. The first is based on Table 3.6 which has the crank as the input. The second is based on Table 3.7 which has θ_3 as the input. The third is based on Table 3.8 which has the slider as the input. The first routine is called i*sldcrkc.m*, the second is i*sldcrkco.m*, and the third is called i*sldcrks.m*. The functions will be discussed separately.

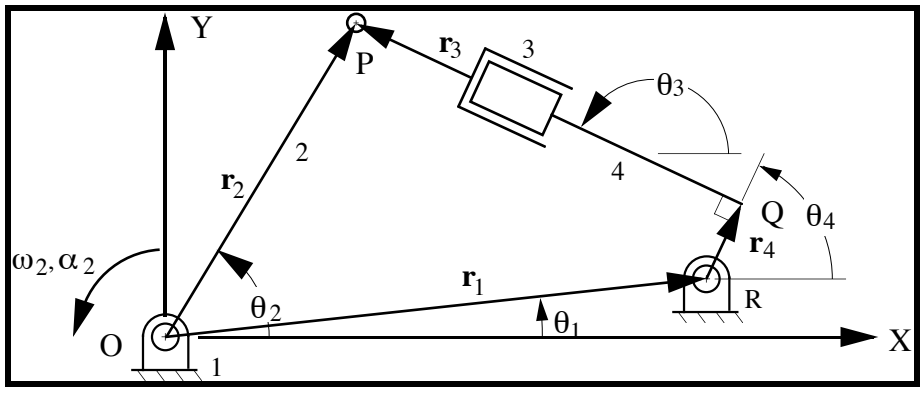


Fig. 3.D.1: Nomenclature for MATLAB Inverted Slider-Crank Function Routines.

3.D.1 Slider-Crank Routine when the Crank Is Driver (*isldcrkc.m*)

The function i*sldcrkc.m* analyzes an inverted slider crank mechanism when the crank is the driving link. The initial statement in the function is:

```
function [values] = isldcrkc(r1,r2,r4,theta2,td2,tdd2,theta1,flag)
```

The input values are:

```
r1      = length of vector 1 (frame)
r2      = length of vector 2 (crank)
r4      = length of vector 4 (slider offset)
theta2  = crank angle (degrees)
td2     = crank angular velocity (rad/sec)
tdd2    = crank angular acceleration (rad/sec^2)
theta1  = angle from frame x axis to line through frame pivots (degrees).
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.  If flag = 2, both a position and velocity analysis is
          conducted.  If flag = 3, a position, velocity, and acceleration
          analysis is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of slider pin (point P)
values (27:28) = x,y components of position of offset point (point Q)
values (29:30) = x,y components of velocity of slider pin (point P)
values (31:32) = x,y components of velocity of offset point (point Q)
values (33:34) = x,y components of acceleration of slider pin (point P)
values (35:36) = x,y components of acceleration of offset point (point Q)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

3.D.1.1 Source Code for Sample m-file (Examples 3.6) Using *isldcrkc.m*

A copy of the source code for a test program is given below. The results are given in Table 3.D.1.

```
% Solution to Example 3.6

clear;
r1=20;
r2=8.5;
r4=0;
theta1=0;
theta2=60;
td2=2.5;
tdd2=0;
flag=3;

values = isldcrkc(r1,r2,r4,theta2,td2,tdd2,theta1,flag);
```

Table 3.D.1: Results for Example 3.6 using *isldcrks.m*

```
r    =   20.0000    8.5000   17.3853         0
th   =         0   60.0000  154.9496   64.9496
rd   =         0         0  -21.1708         0
thd  =         0   -2.5000    0.1055         0
rdd  =         0         0    4.7770         0
thdd =         0         0    3.3012         0
rp   =    4.2500    7.3612
rq   =        20         0
rdp  =   18.4030  -10.6250
rdq  =         0         0
rddp =  -26.5625  -46.0076
rddq =         0         0
```

3.D.2 Slider-Crank Routine when the Coupler Angle Is Driver (*isldcrkco.m*)

The function i*sldcrkco.m* analyzes an inverted slider crank mechanism when the coupler angle is the driver. The initial statement in the function is:

```
function [values] = isldcrkco(r1,r2,r4,theta3,td3,tdd3,beta,theta1,flag)
```

The input values are:

```
r1      = length of vector 1 (frame)
r2      = length of vector 2 (crank)
r4      = length of vector 4 (slider offset)
theta3  = crank angle (degrees)
td3     = crank angular velocity (rad/sec)
tdd3    = crank angular acceleration (rad/sec^2)
beta    = +1 or -1. Identifies crank configuration (and assembly mode)
theta1  = angle from frame x axis to line through frame pivots (degrees).
flag    = analysis flag.  If flag = 1, only a position analysis is conducted.
          If flag = 2, both a position and velocity analysis is conducted.
          If flag = 3, a position, velocity, and acceleration analysis
                    is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of slider pin (point P)
values (27:28) = x,y components of position of offset point (point Q)
values (29:30) = x,y components of velocity of slider pin (point P)
values (31:32) = x,y components of velocity of offset point (point Q)
values (33:34) = x,y components of acceleration of slider pin (point P)
values (35:36) = x,y components of acceleration of offset point (point Q)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

3.D.2.1 Source Code for Sample m-file (Examples 3.6) Using *isldcrkco.m*

A copy of the source code for a test program is given below. The results are identical to those given in Table 3.D.1.

% Solution to inverse of Example 3.6 with theta3 as input

```
clear all;
r1=20;
r2=8.5;
r4=0;
theta1=0;
theta3=154.949611;
td3=0.105459;
tdd3=3.301183;
beta=-1;
flag=3;

values = isldcrkco(r1,r2,r4,theta3,td3,tdd3,beta,theta1,flag);
```

3.D.3 Inverted Slider-Crank Routine when the Slider Is Driver (*isldcrks.m*)

The function i*sldcrks.m* analyzes an inverted slider crank mechanism when the crank is the driving link. The initial statement in the function is:

```
function [values] = isldcrks(r1,r2,r3,rd3,rdd3,r4,beta,theta1,flag)
```

The input values are:

```
r1      = length of vector 1 (frame)
r2      = length of vector 2 (crank)
r3      = length of vector 3 (slider)
rd3     = slider velocity (length/sec)
rdd3    = slider acceleration (length/sec^2)
r4      = length of vector 4 (slider offset)
beta    = +1 or -1.  Identifies assembly mode
theta1  = angle from frame x axis to line through frame pivots (degrees).
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.  If flag = 2, both a position and velocity analysis is
          conducted.  If flag = 3, a position, velocity, and acceleration
          analysis is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of slider pin (point P)
values (27:28) = x,y components of position of offset point (point Q)
values (29:30) = x,y components of velocity of slider pin (point P)
values (31:32) = x,y components of velocity of offset point (point Q)
values (33:34) = x,y components of acceleration of slider pin (point P)
values (35:36) = x,y components of acceleration of offset point (point Q)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

3.D.3.1 Source Code for Sample m-file (Example 3.6 with R3 as input) using *isldcrks.m*

A copy of the source code for a test program is given below. The results are identical to those given in Table 3.D.1.

% Solution to inverse of Example 3.6 with R3 as input

```
clear all;
r1=20;
r2=8.5;
r3=17.385339;
r4=0;
theta1=0;
rd3=-21.170758;
rdd3=4.776957;
beta=1;
flag=3;

values = isldcrks(r1,r2,r3,rd3,rdd3,r4,beta,theta1,flag);
```

3.E MATLAB Functions For RPRP Mechanism Analysis

The MATLAB function files are based on the nomenclature shown in Fig. 3.30 which is repeated here as Fig. 3.E.1 Three routines are given. The first is based on Table 3.9 for the crank as the input. The second is based on Table 3.10 for the slider on the frame as the input. The third is based on Table 3.11 for the slider on the coupler as the input. The first routine is called *rprpc.m*, the second is called *rprps.m*, and the second is called *rprpco.m*. The functions will be discussed separately.

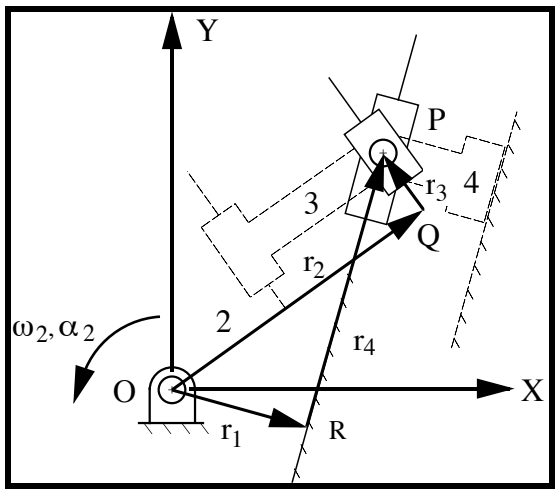


Fig. 3.E.1: Nomenclature for MATLAB RPRP Function Routines.

3.E.1 RPRP Mechanism Analysis Routine when the Crank Is Driver (*rprpc.m*)

The function *rprpc.m* analyzes an RPRP mechanism when the crank is the driving link. The initial statement in the function is:

```
function [values] = rprpc(r1,r2,theta1,theta2,td2,tdd2,flag)
```

The input values are:

```
   r1      = length of vector 1 (frame offset)
   r2      = length of vector 4 (crank offset)
   theta1  = angle from frame x axis to vector r1 (degrees).
   theta2  = crank angle (degrees)
   td2     = crank angular velocity (rad/sec)
   tdd2    = crank angular acceleration (rad/sec^2)
   flag    = analysis flag.  If flag = 1, only a position analysis is
             conducted.  If flag = 2, both a position and velocity analysis
             are conducted.  If flag = 3, position, velocity, and acceleration
             analyses are conducted.

The results are returned in the vector "values".  The answers are stored in values
according to the following:

   values (1:4)   = vector lengths
   values (5:8)   = vector angles (degrees)
   values (9:12)  = derivatives of vector lengths
   values (13:16) = derivatives of vector angles (rad/sec)
   values (17:20) = second derivatives of vector lengths
   values (21:24) = second derivatives of vector angles (rad/sec^2)
   values (25:26) = x,y components of position of slider pin (point P)
   values (27:28) = x,y components of position of slider offset point (point
                    Q)
   values (29:30) = x,y components of velocity of slider pin (point P)
   values (31:32) = x,y components of velocity of slider offset point (point
                    Q)
   values (33:34) = x,y components of acceleration of slider pin (point P)
   values (35:36) = x,y components of acceleration of slider offset point
                    (point Q)
```

3.E.1.1 Source Code for Sample m-file (Example 3.7) using *rprpc.m*

A copy of the source code for a test program is given below. The results are given in Table 3.E.1.

```
% Solution to Example 3.7

clear;
r1=10;
r2=0;
theta1=90;
theta2=-30;
theta4=180;
td2=10;
tdd2=0;
flag=3;

values = rprpc(r1,r2,theta1,theta2,td2,tdd2,flag);
```

Table 3.E.1: Results for Example 3.7 using *rprpc.m*

```
r    =          10.0000         0   11.5470   -5.7735
th   =               90       -30        60       180
rd   =                0         0  -66.6667  133.3333
thd  =                0        10        10         0
rdd  =                0         0    1924.5   -1539.6
thdd =                0         0         0         0
rp   =           5.7735   10.0000
rq   =           5.7735   10.0000
rdp  =        -133.3333         0
rdq  =                0         0
rddp =           1539.6         0
rddq =                0         0
```

3.E.2 RPRP Mechanism Analysis Routine When the Slider on the Frame Is the Driver (*rprps.m*)

The function *rprps.m* analyzes an RPRP mechanism when the slider (link 4) on the frame is the driving link. The initial statement in the function is:

```
function [values] = rprps(r1,r2,r4,rd4,rdd4,theta1,sigma,flag)
```

The input values are:

```
r1      = length of vector 1 (frame offset)
r2      = length of vector 2 (crank offset)
r4      = length of vector 4 (slider)
rd4     = derivative of length of vector 4
rdd4    = second derivative of length of vector 4
theta1  = angle from frame x axis to vector r1 (degrees).
sigma   = +1 or -1.  Identifies offset configuration
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.  If flag = 2, both a position and velocity analysis
          are conducted.  If flag = 3, position, velocity, and acceleration
          analyses are conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of slider pin (point P)
values (27:28) = x,y components of position of slider offset point (point
                 Q)
values (29:30) = x,y components of velocity of slider pin (point P)
values (31:32) = x,y components of velocity of slider offset point (point
                 Q)
values (33:34) = x,y components of acceleration of slider pin (point P)
values (35:36) = x,y components of acceleration of slider offset point
                 (point Q)
```

```
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot be
                 assembled.
```

3.E.2.1 Source Code for Sample m-file (Example 3.7 with R4 as Input) Using *rprps.m*

A copy of the source code for a test program is given below. The results are essentially the same as those listed in Table 3.E.1.

```
% Solution to Example 3.7 when the slider on the frame is the input.

clear all;
r1=10;
r2=0;
r4=-5.773503;
rd4=133.333333;
rdd4=-1539.600718;
sigma=1;
theta1=90;
td2=10;
tdd2=0;
flag=3;

values = rprps(r1,r2,r4,rd4,rdd4,theta1,sigma,flag);
```

3.E.3 RPRP Mechanism Analysis Routine When the Slider on the Coupler Is the Driver (*rprpsc.m*)

The function *rprpsc.m* analyzes an rprp mechanism when the slider (link 3) on the coupler is the driving link. The initial statement in the function is:

```
function [values] = rprpsc(r1,r2,r3,rd3,rdd3,theta1,sigma,flag)
```

The input values are:

```
r1      = length of vector 1 (frame offset)
r2      = length of vector 2 (crank offset)
r3      = length of vector 3 (slider)
rd3     = derivative of length of vector 3
rdd3    = second derivative of length of vector 3
theta1  = angle from frame x axis to vector r1 (degrees).
sigma   = +1 or -1.  Identifies offset configuration
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.  If flag = 2, both a position and velocity analysis
          are conducted.  If flag = 3, position, velocity, and acceleration
          analyses are conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of slider pin (point P)
values (27:28) = x,y components of position of slider offset point (point
```

```
                  Q)
values (29:30) = x,y components of velocity of slider pin (point P)
values (31:32) = x,y components of velocity of slider offset point (point
                  Q)
values (33:34) = x,y components of acceleration of slider pin (point P)
values (35:36) = x,y components of acceleration of slider offset point
                  (point Q)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot be
                  assembled.
```

3.E.3.1 Source Code for Sample m-file (Example 3.7 with R3 as Input) Using *rprpsc.m*

A copy of the source code for a test program is given below. The results are essentially the same as those listed in Table 3.E.1.

```
% Solution to Example 3.7 when the slider on the coupler is the input.

clear all;
r1=10;
r2=0;
r4=-5.773503;
rd4=133.333333;
rdd4=-1539.600718;
sigma=1;
theta1=90;
td2=10;
tdd2=0;
flag=3;

values = rprpsc(r1,r2,r3,rd3,rdd3,theta1,sigma,flag);
```

3.F MATLAB Functions For RRPP Mechanism Analysis

The MATLAB function files are based on the nomenclature shown in Fig. 3.35 which is repeated here as Fig. 3.F.1 Three routines are given. The first is based on Table 3.12 which has the crank, r_2, as the input. The second is based on Table 3.13 when slider 4 is the input, and the third is based on Table 3.13 when slider 3 is the input. The first routine is called *rrppc.m*, the second is called *rrpps.m*, and the third is called *rrppsc.m* The functions will be discussed separately.

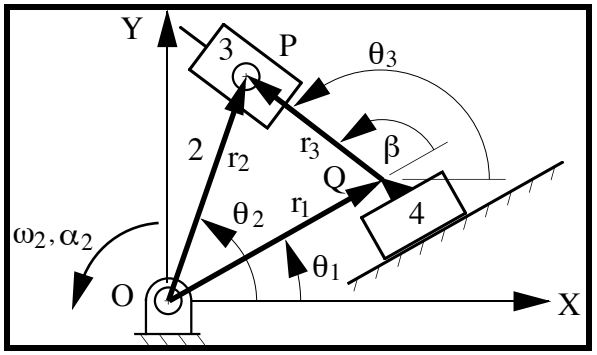


Fig. 3.F.1: Nomenclature for MATLAB RRPP Function Routines.

3.F.1 RRPP Mechanism Analysis Routine when the Crank Is Driver (*rrppc.m*)

The function *rrppc.m* analyzes an RRPP mechanism when the crank is the driving link. The initial statement in the function is:

```
function [values] = rprpc(r1,r2,theta1,theta2,td2,tdd2,flag)
```

The input values are:

```
r2      = length of vector 2 (crank)
theta1  = angle from frame x axis to vector r1 (degrees).
theta2  = crank angle (degrees)
td2     = crank angular velocity (rad/sec)
tdd2    = crank angular acceleration (rad/sec^2)
beta    = angle between slider line on coupler and slider line on
          frame (degrees)
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.  If flag = 2, both a position and velocity analysis
          are conducted.  If flag = 3, position, velocity, and acceleration
          analyses are conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of point P
values (27:28) = x,y components of position of point Q
values (29:30) = x,y components of velocity of point P
values (31:32) = x,y components of velocity of point Q
values (33:34) = x,y components of acceleration of point P
values (35:36) = x,y components of acceleration of point Q
```

3.F.1.1 Source Code for Sample m-file (Example 3.8) Using *rrppc.m*

A copy of the source code for a test program is given below. The results are given in Table 3.F.1.

```
% Solution to Example 3.8 with crank driving

clear;
r2=2;
theta1=0;
theta2=60;
beta=90;
td2=1;
tdd2=0;
flag=3;

values = rrppc(r2,theta1,theta2,td2,tdd2,beta,flag);
```

Table 3.F.1: Results for Example 3.8 using *rrppc.m*

```
r    =    1.0000    2.0000    1.7321
th   =         0        60        90
rd   =   -1.7321         0    1.0000
thd  =         0         1         0
rdd  =   -1.0000         0   -1.7321
thdd =         0         0         0
rp   =    1.0000    1.7321
rq   =         1         0
rdp  =   -1.7321    1.0000
rdq  =   -0.8660   -1.5000
rddp =   -1.0000   -1.7321
rddq =        -1         0
```

3.F.2 RRPP Mechanism Analysis Routine when the Slider on the Frame Is the Driver (*rrpps.m*)

The function *rrppc.m* analyzes an rrpp mechanism when the crank is the driving link. The initial statement in the function is:

```
function [values] = rrpps(r1,r2,rd1,rdd1,theta1,sigma,beta,flag)
```

The input values are:

```
r1      = length of vector 1 (frame offset)
r2      = length of vector 2 (crank offset)
rd1     = derivative of length of vector 1
rdd1    = second derivative of length of vector 1
theta1  = angle from frame x axis to vector r1 (degrees).
sigma   = +1 or -1.  Identifies offset configuration
beta    = angle between slider line on coupler and slider line on frame
          (degrees)
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.
          If flag = 2, both a position and velocity analysis are conducted.
          If flag = 3, position, velocity, and acceleration analyses are
          conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of slider pin (point P)
values (27:28) = x,y components of position of slider offset point (point
                 Q)
values (29:30) = x,y components of velocity of slider pin (point P)
values (31:32) = x,y components of velocity of slider offset point (point
                 Q)
values (33:34) = x,y components of acceleration of slider pin (point P)
values (35:36) = x,y components of acceleration of slider offset point
                 (point Q)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
```

```
                    assembled.
```

3.F.2.1 Source Code for Sample m-file (Example 3.8 with R1 as input) Using *rrpps.m*

A copy of the source code for a test program is given below. The results are essentially the same as those given in Table 3.F.1.

```
% Solution to Example 3.8 when the slider on the frame is the driver

clear;
r1=1;
r2=2;
rd1= -1.732051;
rdd1=-1.000000;
sigma=1;
beta=90;
theta1=0;
flag=3;

values = rrpps(r1,r2,rd1,rdd1,theta1,sigma,beta,flag);
```

3.F.3 RRPP Mechanism Analysis Routine when the Slider on the Coupler Is the Driver (*rrppsc.m*)

The function *rrppc.m* analyzes an RRPP mechanism when the slider on the coupler is the driving link. The initial statement in the function is:

```
function [values] = rrppsc(r3,r2,rd3,rdd3,theta1,sigma,beta,flag)
```

The input values are:

```
r3      = length of vector 3 (frame offset)
r2      = length of vector 2 (crank offset)
rd3     = derivative of length of vector 3
rdd3    = second derivative of length of vector 3
theta1  = angle from frame x axis to vector r1 (degrees).
sigma   = +1 or -1.  Identifies offset configuration
beta    = angle between slider line on coupler and slider line on frame
          (degrees)
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.
          If flag = 2, both a position and velocity analysis are conducted.
          If flag = 3, position, velocity, and acceleration analyses are
          conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of slider pin (point P)
values (27:28) = x,y components of position of slider offset point (point
                 Q)
values (29:30) = x,y components of velocity of slider pin (point P)
values (31:32) = x,y components of velocity of slider offset point (point
```

```
                  Q)
values (33:34) = x,y components of acceleration of slider pin (point P)
values (35:36) = x,y components of acceleration of slider offset point
                  (point Q)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                  assembled.
```

3.F.3.1 Source Code for Sample m-file (Example 3.8 with R1 as input) Using *rrpps.m*

A copy of the source code for a test program is given below. The results are essentially the same as those given in Table 3.F.1.

```
% Solution to Example 3.8 when the slider on the coupler is the driver

clear;
r1=1;
r2=2;
rd1= -1.732051;
rdd1=-1.000000;
sigma=1;
beta=90;
theta1=0;
flag=3;

values = rrppsc(r1,r2,rd1,rdd1,theta1,sigma,beta,flag);
```

3.G MATLAB Functions For Elliptic Trammel Analysis

The MATLAB function files are based on the nomenclature shown in Fig. 3.40 which is repeated here as Fig. 3.G.1 Two routines are given. The first is based on Table 3.14 which has the coupler as the input. The second is based on Table 3.15 which has slider 2 as the input. The first routine is called *prrpc.m* and the second is called *prrps.m*. The functions will be discussed separately.

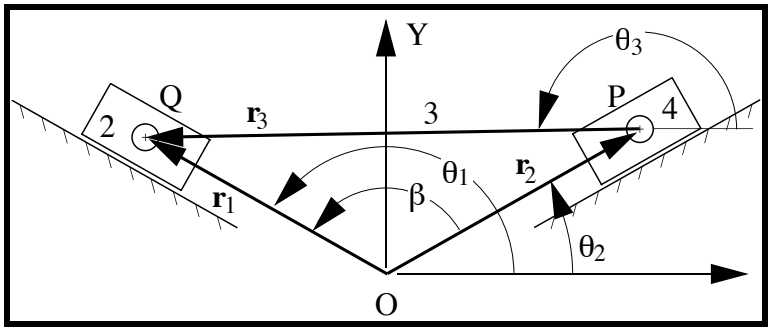


Fig. 3.G.1: Nomenclature for MATLAB Elliptic Trammel Function Routines.

3.G.1 RRPP Mechanism Analysis Routine when the Crank Is Driver (*prrpc.m*)

The function *prrp.m* analyzes an PRRP mechanism when the crank is the driving link. The initial statement in the function is:

```
function [values] = prrpc(r3,theta2,theta3,td3,tdd3,beta,flag)
```

The input values are:

```
r3      = length of coupler
theta3  = angle from frame x axis to vector r3 (degrees)
td3     = angular velocity of coupler (rad/sec)
tdd3    = angular acceleration of coupler (rad/sec^2)
theta2  = angle from frame x axis to vector r2 (degrees).
beta    = angle from slider line 2 to slider line 1 (degrees)
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.
          If flag = 2, both a position and velocity analysis are conducted.
          If flag = 3, position, velocity, and acceleration analyses are
          conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of slider pin (point P)
values (27:28) = x,y components of position of slider offset point (point
                 Q)
values (29:30) = x,y components of velocity of slider pin (point P)
values (31:32) = x,y components of velocity of slider offset point (point
                 Q)
values (33:34) = x,y components of acceleration of slider pin (point P)
values (35:36) = x,y components of acceleration of slider offset point
                 (point Q)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

3.G.1.1 Source Code for Sample m-file (Example 3.9) using *rpprc.m*.

A copy of the source code for a test program is given below. This program animates the linkage through 360° of coupler rotation and traces out the elliptical coupler path for point R. Numerical results are given in Fig. 3.G.1.

```
% Solution to Example 3.9 with coupler driving.  This routine calls rpprc in a
% for loop for 1-deg increments of theta3.

clear all;
r3=10;
rqr=20;
theta2=0;
beta=90;
td3=10;
tdd3=0;
flag=3;
fact=pi/180;
i=0;

for th3=0:1:360
    i=i+1;
    theta3(i)=th3;
%
% Solve the problem for 1 degree increments of theta3.
%
    values = prrpc(r3,theta2,theta3(i),td3,tdd3,beta,flag);

    rp(1:2)=values(25:26);
```

```
    rq(1:2)=values(27:28);
    rdp(1:2)=values(29:30);
    rddp(1:2)=values(33:34);
    rrx(i)=rq(1)+rqr*cos((theta3(i)+180)*fact);
    rry(i)=rq(2)+rqr*sin((theta3(i)+180)*fact);
    rpx(i)=rp(1);
    rpy(i)=rp(2);
    rqx(i)=rq(1);
    rqy(i)=rq(2);
    drp(i)=rdp(1);
    ddrp(i)=rddp(1);

end
itotal=i;
tt=2;

%plot the results.

clf
ans='y';
disp('   ')
disp('         Elliptic Trammel Mechamism')

% Define velocity axes.

h2=axes('position', [0.6 .55 .38 .35],'box', 'on','xcolor', 'k',...
        'ycolor', 'k');   %axis location
set(gcf,'color', 'w');;
xlabel('theta3 (deg)','color', 'k')
ylabel('velocity of point P (cm/s)','color', 'k')
VelP=line('xdata', [], 'ydata', [],  'erasemode', 'none','visible', 'on',...
          'color', 'r');
hbead1=line('xdata',[],'ydata',[],'marker','o','erase','xor','markersize',6);

% Define acceleration axes.

h3=axes('position', [0.6 .1 .38 .35],'box', 'on','xcolor', 'k','ycolor','k');
set(gcf,'color', 'w');;
xlabel('theta3 (deg)','color', 'k')
ylabel('acceleration of point P (cm/s)','color', 'k')
AccP=line('xdata', [], 'ydata', [],  'erasemode', 'none', 'color', 'r');
hbead2=line('xdata',[],'ydata',[],'marker','o','erase','xor','markersize',6);

% Define position axes.

h1=axes('position', [0.1 .55 .38 .35],'box', 'on', 'xcolor', 'k','ycolor', 'k');
set(gcf,'color', 'w');;
coupler=line('xdata', [], 'ydata' ,[], 'linewidth' ,3, 'erasemode', 'xor',...
             'color', 'k');
couplerpoint=line('xdata',[],'ydata',[],'linewidth',1,'linestyle', '--',...
             'erasemode','none','color',[1,0,0]);
blockp=line('xdata', [], 'ydata',[],'erase','xor', 'color', 'k');
blockq=line('xdata', [], 'ydata',[],'erase','xor', 'color', 'k');
framelinep=line('xdata', [], 'ydata', [], 'erasemode', 'none', 'color', 'k');
framelineq=line('xdata', [], 'ydata', [], 'erasemode', 'none', 'color', 'k');

% Identify coordinates for frame line at P.

x0=-1.3*r3;
y0=-(r3/12)*1.05;
length=-2*x0;
angle=0;
ndash=20;
flag=1;
coord=frameline(length,x0,y0,ndash,angle,flag);
```

```
npoints=3*ndash;
for j=1:1:npoints
    xfp(j)=coord(j,1);
    yfp(j)=coord(j,2);
end

% Identify coordinates for frame line at Q.

y0=-1.3*r3;
x0=-(r3/12)*1.05;
length=-2*y0;
angle=90;
flag=-1;
coord=frameline(length,x0,y0,ndash,angle,flag);
for j=1:1:npoints
    xfq(j)=coord(j,1);
    yfq(j)=coord(j,2);
end

% Identify box for slider at point P.

length=r3/4;
width=r3/6;

% Set the axis limits.

    axes(h2);
%set axis limits for velocity of P

    axis([min(theta3) max(theta3) (1.05)*min(drp) (1.05)*max(drp)]);

%set axis limits for acceleration of P
    axes(h3);
    axis([min(theta3) max(theta3) (1.05)*min(ddrp) (1.05)*max(ddrp)]);

%set axis limits for the geometry
    axes(h1);
    axis([(1.05)*min(rrx) (1.05)*max(rrx) (0.9)*min(rry) (1.05)*max(rry)]);
    axis('equal')

% Draw curves for each plot.

    set(VelP,'xdata', theta3,'ydata', drp);
    set(AccP,'xdata', theta3,'ydata', ddrp);
    set(couplerpoint,'xdata', rrx,'ydata',rry);
    set(framelinep,'xdata', xfp,'ydata',yfp);
    set(framelineq,'xdata', xfq,'ydata',yfq);

% Animate the results.

length=r3/4;
width=r3/6;
r=r3/10;
ninc=20;
npoints=ninc+1;
while ans=='y'
    for j=1:1:tt;
        for i=1:1:itotal;

            % Draw block at P

            set(framelinep,'xdata', xfp,'ydata',yfp);
            set(framelineq,'xdata', xfq,'ydata',yfq);
            flag=0;
            angle=0;
```

```
            x0=rpx(i);
            y0=rpy(i);
            coord=rect(length,width,x0,y0,angle,flag);
            for j=1:1:5
                xcoord(j)=coord(j,1);
                ycoord(j)=coord(j,2);
            end
            set(blockp,'xdata',xcoord,'ydata',ycoord);

            % Draw block at Q

            flag=0;
            x0=rqx(i);
            y0=rqy(i);
            angle=90;
            coord=rect(length,width,x0,y0,angle,flag);
            for j=1:1:5
                xcoord(j)=coord(j,1);
                ycoord(j)=coord(j,2);
            end
            set(blockq,'xdata',xcoord,'ydata',ycoord);

            % Draw coupler line

            set(coupler,'xdata',[rqx(i) rrx(i)],'ydata',[rqy(i) rry(i)]);
            set(hbead1,'xdata',theta3(i),'ydata',drp(i));
            set(hbead2,'xdata',theta3(i),'ydata',ddrp(i));
            drawnow;   %flush the draw buffer
        end
        i=i+1; %increments for loop
    end    %ends while loop

% Draw in final position
i=13;

    % Draw block at P

    flag=0;
    angle=0;
    x0=rpx(i);
    y0=rpy(i);
    coord=rect(length,width,x0,y0,angle,flag);
    for j=1:1:5
        xcoord(j)=coord(j,1);
        ycoord(j)=coord(j,2);
    end
    set(blockp,'xdata',xcoord,'ydata',ycoord);

    % Draw block at Q

    flag=0;
    x0=rqx(i);
    y0=rqy(i);
    angle=90;
    coord=rect(length,width,x0,y0,angle,flag);
    for j=1:1:5
        xcoord(j)=coord(j,1);
        ycoord(j)=coord(j,2);
    end
    set(blockq,'xdata',xcoord,'ydata',ycoord);

    % Draw coupler line

    set(coupler,'xdata',[rqx(i) rrx(i)],'ydata',[rqy(i) rry(i)]);
    set(hbead1,'xdata',theta3(i),'ydata',drp(i));
```

```
    set(hbead2,'xdata',theta3(i),'ydata',ddrp(i));
    drawnow;   %flush the draw buffer

%   Ask if the animation should be repeated
    ans=input('Repeat animation? y/n [y]: ', 's');
    if isempty(ans);
        ans='y';
    end
end

% Computer the results for one position.

r3=10;
sigma=1;
beta=-90;
theta2=90;
flag=3;
beta=-90;
td3=10;
tdd3=0;
flag=3;
fact=pi/180;
theta3(1)=-30;

values = prrpc(r3,theta2,theta3(1),td3,tdd3,beta,flag);

r(1:4)=values(1:4)
th(1:4)=values(5:8)
rd(1:4)=values(9:12)
thd(1:4)=values(13:16)
rdd(1:4)=values(17:20)
thdd(1:4)=values(21:24)
rp(1:2)=values(25:26)
rq(1:2)=values(27:28)
rdp(1:2)=values(29:30)
rdq(1:2)=values(31:32)
rddp(1:2)=values(33:34)
rddq(1:2)=values(35:36)

fprintf('example_3p9.dat','%10.6f %10.6f %10.6f %10.6f \n',values)
```

Table 3.G.1: Numerical results for Example 3.9 when theta3 is -30˚ using *prrpc.m*. The graphical results are shown in Fig. 3.44

```
r    =    8.6603    5.0000   10.0000
th   =         0        90       -30
rd   =   50.0000  -86.6025         0
thd  =         0         0        10
rdd  = -866.0254 -500.0000         0
thdd =         0         0         0
rp   =         0    5.0000
rq   =    8.6603         0
rdp  =         0  -86.6025
rdq  =   50.0000         0
rddp =         0 -500.0000
rddq = -866.0254         0
```

3.G.2 PRRP Mechanism Analysis Routine when the Slider at Link 2 Is the Driver (*prrps.m*)

The function *prrps.m* analyzes an PRRP mechanism when the slider at link 2 is the driving link. The initial statement in the function is:

```
function [values] = prrps(r1,rd1,rdd1,r3,theta2,sigma,beta,flag)
```

The input values are:

```
r1      = length of vector 1
rd1     = derivative of length of vector 1
rdd1    = second derivative of length of vector 1
r3      = length of coupler
theta2  = angle from frame x axis to vector r2 (degrees).
sigma   = +1 or -1.  Identifies offset configuration
beta    = angle between slider line 2 and slider line 1 (degrees)
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.
          If flag = 2, both a position and velocity analysis are conducted.
          If flag = 3, position, velocity, and acceleration analyses
          are conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of point P
values (27:28) = x,y components of position of point Q
values (29:30) = x,y components of velocity of point P
values (31:32) = x,y components of velocity of point Q
values (33:34) = x,y components of acceleration of point P
values (35:36) = x,y components of acceleration of point Q
```

```
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

3.G.2.1 Source Code for Sample m-file (Example 3.9 with R1 as the Input) using *rpprs.m*.

A copy of the source code for a test program is given below. The results are essentially the same as those given in Table 3.G.1.

```
% Solution to Example 3.9 when a slider (link 2) is the driver.  The position
% analyzed corresponds to Theta3 = -30 deg.

clear;
r3=10;
r1=8.66025;
rd1=50.0000;
rdd1= -866.0254;
sigma=1;
beta=-90;
theta2=90;
flag=3;
values = prrps(r1,rd1,rdd1,r3,theta2,sigma,beta,flag);
```

3.H MATLAB Utility Routines For Rectangle, Frameline, Circle, and Bushing

This appendix contains the MATLAB files to define the points necessary to draw a rectangle, a hatched line (for representing the frame), a circle, and a bushing attached to the frame. The routines do not actually draw the items but they return the points which will facilitate drawing them.

3.H.1 Rectangle Routine

The rectangle routine return the points corresponding to the corners of a rectangle given the coordinates of either the center or bottom left-hand corner and the inclination angle. The principle quantities are shown in Fig. 3.H.1. The routine is called *rectangle.m*, and the initial statement in the function is given below. The function determines the coordinates of a series of points defining a rectangle. The number of points stored is 5, but the first and last points are the same.

```
function [coords] = rectangle(length,height,x0,y0,theta,flag)
```

The input values are:

```
length  = length of rectangle
height  = height of rectangle
x0      = x coordinate of center or bottom left corner of rectangle
y0      = y coordinate of center or bottom left corner of rectangle
angle   = rotation angle relative to the horizontal x axis (degrees)
flag    = flag indicating if the rotation and (X0, y0) are defined
          relative to the center (flag = 0) or to the bottom left hand
          corner of the rectangle.
```

The results are returned in the vector "coords". The answers are stored in values according to the following:

```
values (i,1) = x coordinates of rectangle
values (i,2) = y coordinates of rectangle
```

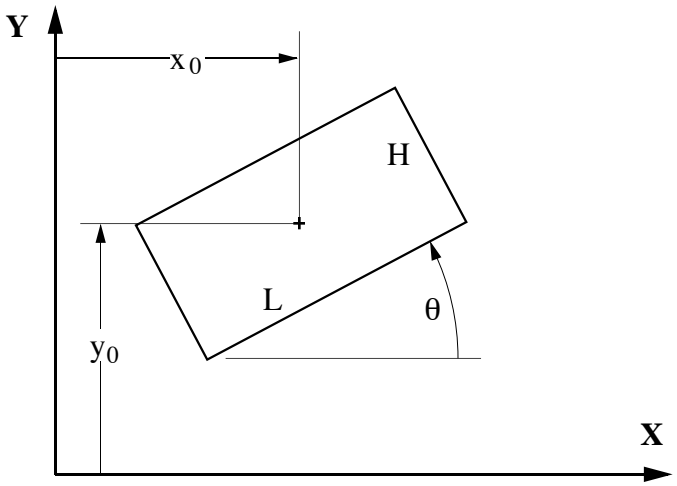


Fig. 3.H.1: Nomenclature for rectangle routine.

3.H.2 Frameline Routine

The frameline routine return the points corresponding to the points needed to draw a hatched line which can be used to represent a frame. The line can be drawn at any angle, and the hatched lines can be either above or below the line. The hatches are drawn at 45˚ to the base line. The routine is called *frameline.m*, and the initial statement in the function is given below.

```
function [coords] = frameline(length,x0,y0,ndash,theta,flag)
```

The input values are:

```
length  = radius of frame line
x0      = x coordinate of left end of line
y0      = y coordinate of left end of line
ndash   = number of dashes at -45 degrees to base
angle   = rotation angle relative to the horizontal x axis (degrees)
flag    = slide flag.  If flag = 1, the dashes are drawn on the bottom
          of the line.  If flag = -1, they are drawn on the top.
```

The results are returned in the vector "coords". The answers are stored in values according to the following:

```
coords (i,1) = x coordinates of frame line
coords (i,2) = y coordinates of frame line
```

3.H.3 Circle Routine

The circle routine determines the coordinates of a series of points around a circle of a given radius and center location. The number of points stored is ninc+1. The routine is called *circle.m*, and the initial statement in the function is given below.

```
function [coords] = circle(r,x0,y0,ninc)
```

The input values are:

```
r       = radius of circle
```

```
x0      = x coordinate of center of circle
y0      = y coordinate of center of circle
ninc    = number of increments into which circle is divided
```

The results are returned in the vector "coords". The answers are stored in values according to the following:

```
coords (i,1) = x coordinate of points on circle
coords (i,2) = y coordinate of points on circle
```

3.H.4 Bushing Routine

The bushing routine return the points which define a bushing for a mechanism. This function determines the coordinates of a series of points defining the outside of a bushing. The radius of the pin is r. The number of points stored is npoints where npoints = ninc+16. The pin is not computed. It should be determined using the function "circle". The bushing can be rotated at any angle. The based is represented by four hatch lines. The principle quantities are shown in Fig. 3.H.2. The routine is called *bushing.m*, and the initial statement in the function is given below.

```
function [coords] = bushing(r,x0,y0,ninc,theta)
```

The input values are:

```
r       = radius of circle
x0      = x coordinate of center of circle
y0      = y coordinate of center of circle
ninc    = number of increments into which the semicircle is divided
angle   = rotation angle relative to the horizontal x axis (degrees)
```

The results are returned in the vector "coords". The answers are stored in values according to the following:

```
coords (i,1) = x coordinates of bushing
coords (i,2) = y coordinates of bushing
```

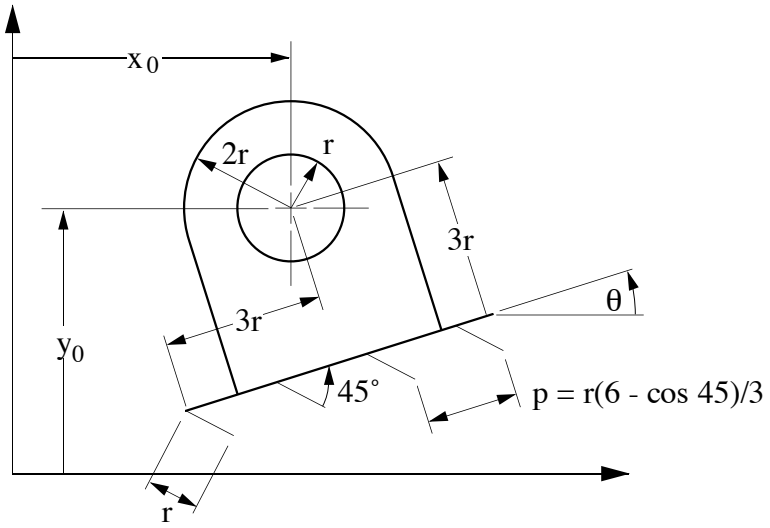


Fig. 3.H.2: Nomenclature for bushing routine.

3.I MATLAB Functions For RPPR Mechanism Analysis

The MATLAB function files are based on the nomenclature shown in Fig. 3.44 which is repeated here as Fig. 3.I.1 Two routines are given. The first is based on Table 3.16 which has the crank, r_2, as the input. The second is based on Table 3.17 when slider 3 is the input. The first routine is called *rpprc.m*, and the second is called *rpprs.m*. The functions will be discussed separately.

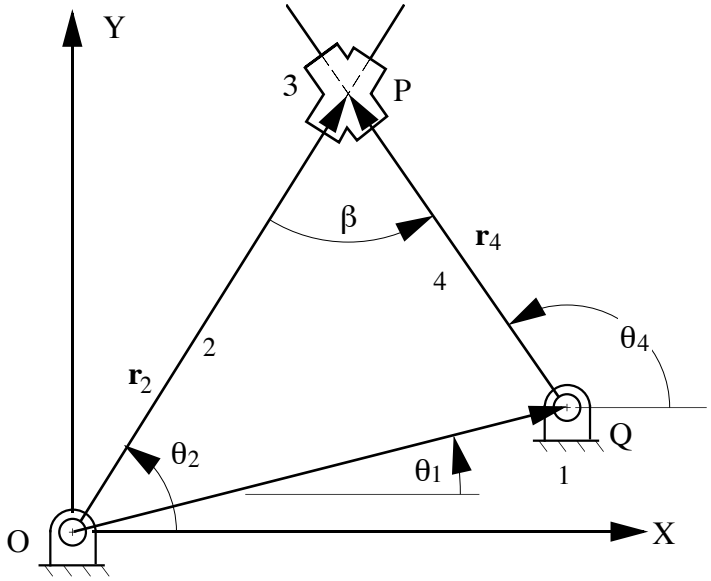


Fig. 3.I.1: Nomenclature for MATLAB RPPR Function Routines

3.I.1 RPPR Mechanism Analysis Routine when the Crank Is Driver (*rpprc.m*)

The function *rrppc.m* analyzes an RPPR mechanism when the crank is the driving link. The initial statement in the function is:

```
function [values] = rpprc(r1,theta1,theta2,td2,tdd2,beta,flag)
```

The input values are:

```
r1      = length of frame
theta1  = angle from frame x axis to vector r1 (degrees)
theta2  = angle from frame x axis to vector r2 (degrees)
td3     = angular velocity of link 2 (rad/sec)
tdd4    = angular acceleration of link 2 (rad/sec^2)
beta    = constant angle from r2 to r4 (degrees)
flag    = analysis flag.  If flag = 1, only a position analysis is conducted.
          If flag = 2, both a position and velocity analysis are conducted.
          If flag = 3, position, velocity, and acceleration analyses
                       are conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of point P
values (27:28) = x,y components of position of point Q
values (29:30) = x,y components of velocity of point P2
values (31:32) = x,y components of velocity of point P3
values (33:34) = x,y components of acceleration of point P2
values (35:36) = x,y components of acceleration of point P3
```

3.I.1.1 Source Code for Sample m-file (Example 3.10) Using *rpprc.m*

A copy of the source code for a test program is given below. The results are given in Table 3.F.1.

```
% Solution to Example 3.10 with crank (theta2) driving

clear all;
r1=10;
theta1=0;
theta2=60;
beta=45;
td2=10;
tdd2=100;
flag=3;
fact=pi/180;
i=0;
values = rpprc(r1,theta1,theta2,td2,tdd2,beta,flag);
```

Table 3.I.1: Results for Example 3.10 using *rpprc.m*

```
r     =   10.0000   13.6603     0   12.2474
th    =         0         60     0       105
rd    =         0  -36.6025     0   70.7107
thd   =         0         10     0        10
rdd   =         0   -1732.1     0    -517.6
thdd  =         0       100     0       100
rp    =    6.8301   11.8301
rq    =        10         0
rdp2  = -118.3013   68.3013
rdp3  = -136.6025   36.6025
rddp2 =   -1866.0    -500.0
rddp3 =   -2098.1   -2366.0
```

3.I.2 RPPR Mechanism Analysis Routine when the Slider on the Crank Is the Driver (*rpprs.m*)

The function *rpprc.m* analyzes an RPPR mechanism when the slider on link 2 is the driving link. The initial statement in the function is:

```
function [values] = rpprs(r1,theta1,r2,rd2,rdd2,beta,sigma,flag)
```

The input values are:

```
   r1     = length of frame
   theta1 = angle from frame x axis to vector r1 (degrees)
   r2     = length of vector 2
   rd2    = derivative of length of vector 2
   rdd2   = second derivative of length of vector 2
   beta   = constant angle from r2 to r4 (degrees)
   sigma  = +1 or -1.  Identifies assembly mode
   flag   = analysis flag. If flag = 1, only a position analysis is conducted.
            If flag = 2, both a position and velocity analysis are conducted.
            If flag = 3, position, velocity, and acceleration analyses
            are conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
   values (1:4)   = vector lengths
   values (5:8)   = vector angles (degrees)
   values (9:12)  = derivatives of vector lengths
   values (13:16) = derivatives of vector angles (rad/sec)
   values (17:20) = second derivatives of vector lengths
   values (21:24) = second derivatives of vector angles (rad/sec^2)
   values (25:26) = x,y components of position of point P
   values (27:28) = x,y components of position of point Q
   values (29:30) = x,y components of velocity of point P2
   values (31:32) = x,y components of velocity of point P3
   values (33:34) = x,y components of acceleration of point P2

values (35:36) = x,y components of acceleration of point P3
```

3.I.2.1 Source Code for Sample m-file (Example 3.10 with R2 as input) Using *rpprs.m*

A copy of the source code for a test program is given below. The results are essentially the same as those given in Table 3.I.1.

```
% Solution to Example 3.10 when the slider on the crank is the driver

   clear all;
   r1= 10;
   theta1=0;
   r2=13.660254;
   rd2=-36.602540;
   rdd2= -1732.050808;
   sigma=1;
   beta=45;
   flag=3;
   values = rpprs(r1,theta1,r2,rd2,rdd2,beta,sigma,flag);
```

4.0 Programs for Chapter 4: Linkage Design

Chapter 4 includes 10 appendices, which contain a description of the MATLAB function routines developed to illustrate the concepts in this Chapter.

4.A MATLAB Functions for Rigid Body Guidance

The rigid-body guidance procedures have been coded in four MATLAB m-files. The first can be used for the design of four-bar linkages with either the center points or circle points input. The second can be used to design a slider crank mechanism when the slider is the driving link. The third designs a crank slider mechanism where the crank is the driving link. The fourth routine designs an elliptic trammel. In this routine, the coupler is assumed to be the driving link. Most of the three-position design examples provided in this book can be solved with these routines.

In all cases, the positions are identified by the coordinates of a point (a_x, a_y) and the angle θ between the frame x axis and a line on the coupler. The center points, circle points, and sliders can be input through either the keyboard or by using the mouse. The linkage can be animated after the joints have been identified. In this way, it is possible to determine if the linkage must change branch to move through all of the desired positions. If the linkage does not travel through all of the desired positions, there is a change in branch. After the animation is completed, there is an option to change the assembly mode and reconduct the analysis. Using this procedure, it will be obvious that the linkage does travel through the positions missed in the other mode. This option is not included in the elliptic trammel routine because this mechanism has only one branch if it is driven by the coupler (see Table 3.14).

Most of the data may be input either interactively or using a file. In the interactive mode, the programs prompts the user for each item of data. Default values have been included in the program, and these can be selected by simply pressing *return*. The default values are shown in square brackets []. The input data are printed to a data file that can be used in subsequent analyses in the "file-input" mode.

If a data file is available, the user needs only identify that a file input is to be used. The program then prompts for the name of the input file and reads the values for the input variables.

The speed of the animation can be changed in several ways. The first is to increase or reduce the number of positions analyzed. The second way is to perform extraneous calculations in the display loop. This is the procedure used when the option is given to speed/slow up the animation. Another way to slow down the animation significantly is to introduce a "pause". However, the minimum delay is one second, and this is too long for most cases.

The m-files used for the calculations are called *rbg_4bar.m*, *rbg_slidercrank.m*, *rbg_crankslider.m*, and *rbg_el_trammel.m*. Each of the routines uses several other MATLAB routines which are identified. Some of these have already been discussed, and the new ones are described below.

4.A.1 Design Routines for Rigid Body Guidance using a Four-Bar Linkage

This set of functions includes a main routine (*rbg_4bar.m*) and ten function routines. Three of the function routines are *axisadjust.m, circle.m* and *bush.m* which were described in Appendix 3.1. The other six are identified in the following.

4.A.1.1 Assembly Mode Routine (*assemblymode.m*)

The assembly mode routine determines the assembly mode for a four-bar linkage designed using rigid-body guidance techniques by checking the mode in the first position. The initial statement in the program is

```
function mode = assemblymode(r1,r2,r3,r4,Q1,theta, phi)
```

The input variables are:

```
r1, r2, r3, r4 = the link lengths numbered in the standard fashion (r1=frame,
   r2=driver, r3=coupler, r4=output).
Q1    = the frame angle in radians
theta = the driver angle in radians
phi   = the output angle in radians
```

The returned value is

```
mode = the assembly mode (+1 or -1)
```

4.A.1.2 Center Point Routine (*centerpoint.m*)

The center point routine determines the coordinates of the center point given the coordinates of the circle point. The initial statement in the program is

```
function [Values] = centerpoint (ax1, ay1, theta1, ax2, ay2, theta2, ax3, ay3,
theta3, XC, YC)
```

The input variables are:

```
ax1     = x coordinate of coupler coordinate system in position 1
ay1     = y coordinate of coupler coordinate system in position 1
ax2     = x coordinate of coupler coordinate system in position 2
ay2     = y coordinate of coupler coordinate system in position 2
ax3     = x coordinate of coupler coordinate system in position 3
ay3     = y coordinate of coupler coordinate system in position 3
theta1  = angle from frame x axis to coupler x axis in position 1 (degrees)
theta2  = angle from frame x axis to coupler x axis in position 2 (degrees)
theta3  = angle from frame x axis to coupler x axis in position 3 (degrees)
XC      = x coordinate of circle point relative to coupler coordinate system
YC      = y coordinate of circle point relative to coupler coordinate system
```

The center point coordinates are returned in the vector "values". The coordinates are stored in values according to the following:

```
Values(1)   = center point x coordinate relative to frame coordinate system
Values(2)   = center point y coordinate relative to frame coordinate system
Values(3)   = circle point x in position 1 relative to frame system
Values(4)   = circle point y in position 1 relative to frame system
Values(5)   = circle point x in position 2 relative to frame system
Values(6)   = circle point y in position 2 relative to frame system
Values(7)   = circle point x in position 3 relative to frame system
Values(8)   = circle point y in position 3 relative to frame system
```

4.A.1.3 Circle Point Routine (*circlepoint.m*)

The center point routine determines the coordinates of the circle point given the coordinates of the center point. This routine uses the routine *pole.m* to find the center of a circle. The initial statement in the function is

```
function [Values] = circlepoint (ax1,ay1,theta1,ax2,ay2,theta2,ax3,ay3,
                  theta3, Astarx,Astary)
```

The input variables are:

```
ax1      = x coordinate of coupler coordinate system in position 1
ay1      = y coordinate of coupler coordinate system in position 1
ax2      = x coordinate of coupler coordinate system in position 2
ay2      = y coordinate of coupler coordinate system in position 2
ax3      = x coordinate of coupler coordinate system in position 3
ay3      = y coordinate of coupler coordinate system in position 3
theta1   = angle from frame x axis to coupler x axis in position 1(deg.)
theta2   = angle from frame x axis to coupler x axis in position 2 (deg.)
theta3   = angle from frame x axis to coupler x axis in position 3 (deg.)
Astarx   = x coordinate of center point relative to coupler coordinate
           system
Astarx   = y coordinate of center point relative to coupler coordinate
           system
```

The circle point coordinates are returned in the vector "values". The coordinates are stored in "values" according to the following:

```
Values(1) = circle point x in position 1 relative to frame system
Values(2) = circle point y in position 1 relative to frame system
Values(3) = circle point x relative to coupler system
Values(4) = circle point y relative to coupler system
```

4.A.1.4 Circle of Sliders Routine (*cosline.m*)

The circle of sliders routine determines the coordinates of 31 points on the circle of sliders when the radius of the circle is infinite. The initial statement in the function is

function coords = cosline(p12,p13,p23,p23prime)

The input variables are:

```
p12 = a three component vector giving the x,y coordinates of the pole P12
      and a flag indicating if the pole is at infinity.  If p12(3) = 0, the
      pole is not at infinity.  If p12(3) = 1, the pole is at infinity.
p13 = a three component vector giving the x,y coordinates of the pole P13
      and a flag indicating if the pole is at infinity.  If p13(3) = 0, the
      pole is not at infinity.  If p13(3) = 1, the pole is at infinity.
p23 = a three component vector giving the x,y coordinates of the pole P23
      and a flag indicating if the pole is at infinity.  If p23(3) = 0, the
      pole is not at infinity.  If p23(3) = 1, the pole is at infinity.
p23prime = a two component vector giving the x, y coordinates of the image pole
   for P23.
```

The results for the 31 points are returned in the vector "coords". The x components of the point j on the line are stored in `coords(j,1)`, and the y components are stored in `coords(j,2)`.

4.A.1.5 Fourbar Mechanism Analysys Routine (*fourbar.m*)

The fourbar routine analyzes a four-bar linkage when the crank is the driving link. The input values are:

```
r1      = length of vector 1 (frame)
r2      = length of vector 2 (crank)
r3      = length of vector 3 (coupler)
r4      = length of vector 4 (slider offset)
theta2  = crank angle (degrees)
td2     = crank angular velocity (rad/sec)
tdd2    = crank angular acceleration (rad/sec^2)
sigma   = +1 or -1.  Identifies assembly mode
theta1  = angle between line through fixed pivots and frame x axis (degrees)
flag    = analysis flag.  If flag = 1, only a position analysis is conducted.
          If flag = 2, both a position and velocity analysis is conducted.
          If flag = 3, a position, velocity, and acceleration analysis is
          conducted.
```

The results are returned in the vector "values" according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector angles (rad/sec)
values (13:16) = second derivatives of vector angles (rad/sec^2)
values (17:18) = x,y components of position of crank pin (point Q)
values (19:20) = x,y components of position of follower pin (point P)
values (21:22) = x,y components of velocity of crank pin (point Q)
values (23:24) = x,y components of velocity of follower pin (point P)
values (25:26) = x,y components of acceleration of crank pin (point Q)
values (27:28) = x,y components of acceleration of follower pin (point P)
values (29)    = assembly flag.  If values(29) = 0, mechanism cannot
                 assembled.
```

4.A.1.6 Pole Routine (*pole.m*)

The MATLAB function file to find the displacement poles is based on the equations given in Sections 4.2.7.2 - 4.2.7.4. This function computes the coordinates of the pole given two positions of two points on the moving rigid body. The initial statement in the program is

```
function pij = pole(a1,a2,b1,b2)
```

The input variables are:

```
a1 = a two component vector giving the x,y coordinates of the first position of
   point a.
a2 = a two component vector giving the x,y coordinates of the second position of
   point a.
b1 = a two component vector giving the x,y coordinates of the first position of
   point b.
b2 =a two component vector giving the x,y coordinates of the second position of
   point b.
```

The returned vector is

```
pij = a two component vector giving the x,y coordinates of the pole.
```

4.A.1.7 Image Pole Routine (*ipole.m*)

This MATLAB function file computes the coordinates of the image pole p'_{23}. The initial statement in the program is

```
function pij = ipole(p12,p13,p23)
```

The input variables are:

```
p12 = x, y coordinates of pole p12
p13 = x, y coordinates of pole p13
p23 = x, y coordinates of pole p23
```

The coordinates of the image pole p'_{23} are returned in the vector pij.

4.A.1.8 Sample Run using *rbg_4bar.m*

The results from a sample run using the four-bar linkage synthesis program are summarized in Table 4.A.1, and the graphical results are shown in Figs. 4.A.1 and 4.A.2.

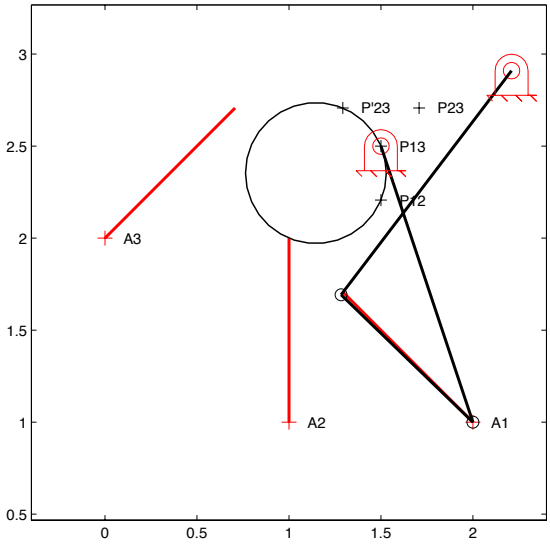


Fig. 4.A.1: Positions specified and basic linkage designed

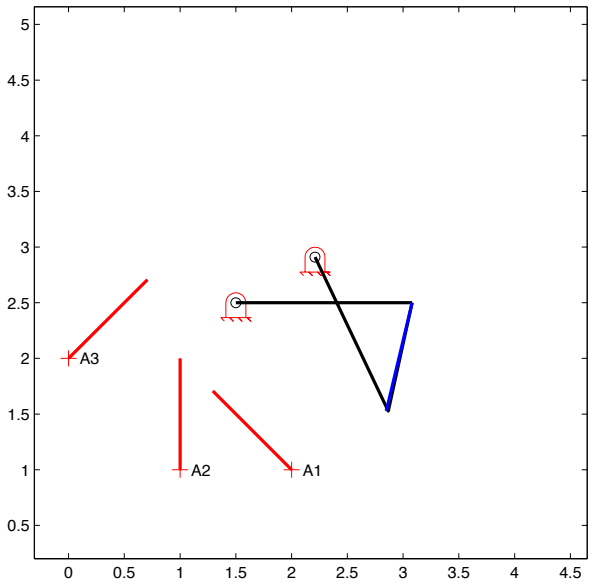


Fig. 4.A.2: Results after animation

Table 4.A.1: Input Data and Numerical Results from *rbg_4bar.m*

```
        Four Bar Linkage Design for Rigid-Body Guidance
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (rgb_4bario.dat): manual.dat
Enter value of ax1 [0]: 2
Enter value of ay1 [0]: 1
Enter value of theta1 (deg) [45]: 135
Enter value of ax2 [3]: 1
Enter value of ay2 [0]: 1
Enter value of theta2 (deg) [135]: 90
Enter value of ax3 [4]: 0
Enter value of ax3 [2]: 2
Enter value of theta3 (deg) [0]: 45

         Input Posiiton Information
   Position     ax(i)      ay(i)     theta(i)
      1         2.000      1.000     135.000
      2         1.000      1.000      90.000
      3         0.000      2.000      45.000

Possible modes of inputting data:
  1 - mouse
  2 - keyboard
Designate mode of input [1] 2

Possible types of points to be input:
  1 - circle point
  2 - center point
Designate type of point to be input [1]: 1

 Input x component of first circle point 2
 Input y component of first circle point 1
```

```
Reinter pivot point? y/n [n]: n

Possible types of points to be input:
  1 - circle point
  2 - center point
Designate type of point to be input [1]: 2

 Input x component of second center point 2.21
 Input y component of second center point 2.91
Reinter pivot point? y/n [n]: n
Reinter pivot points? y/n [n]: n

             Link Lengths
     r1       r2       r3       r4
    0.82     1.58     1.00     1.53

         Center Point Coordinates
   Astarx    Astary    Bstarx    Bstary
    1.50     2.50     2.21     2.91

 Circle Point Coordinates in Position 1
     Ax       Ay       Bx       By
    2.00     1.00     1.28     1.69
Repeat animation? y/n [y]: n
Change assembly mode and repeat analysis? y/n [y]: n
```

4.A.2 Design Routines for Rigid Body Guidance using a Slider-Crank Linkage

This set of functions includes a main routine (*rbg_slidercrank.m*) and thirteen function routines. Eight of the function routines are *axisadjust.m, circle.m*, *bush.m, pole.m, ipole.m, cosline.m, circlepoint.m,* and *centerpoint.m* which were described previously. The other four are identified in the following.

4.A.2.1 Assembly Mode Routine (*assemblymode_sc.m*)

The assembly mode routine determines the assembly mode for a slider-crank linkage designed using rigid-body guidance techniques by checking the mode in the first position. The initial statement in the program is

```
function mode = assemblymode_sc(r2,r3,bx,by,Q1,Q2)
```

The input variables are:

```
r2, r3 = the link lengths numbered in the standard fashion (r2=driver,
         r3=coupler).
bx     = slider x coordinate relative to frame
by     = slider y coordinate relative to frame
Q1     = slide angle in degrees
Q2     = driver angle in degrees
```

The returned value is

```
mode = the assembly mode (+1 or -1)
```

4.A.2.2 Slider Point Routine (*sliderpoint.m*)

The slider point routine determines the coordinates of the slider point in three positions relative to the frame given the coordinates of the slider point relative to the coupler system. The coordinates of the slider line are

also given. The initial statement in the program is

```
function [Values] = sliderpoint(ax1, ay1, theta1, ax2, ay2, theta2, ax3, ay3,
theta3, Xn, Yn)
```

The input variables are:

```
ax1    = x coordinate of coupler coordinate system in position 1
ay1    = y coordinate of coupler coordinate system in position 1
ax2    = x coordinate of coupler coordinate system in position 2
ay2    = y coordinate of coupler coordinate system in position 2
ax3    = x coordinate of coupler coordinate system in position 3
ay3    = y coordinate of coupler coordinate system in position 3
theta1 = angle from frame x axis to coupler x axis in position 1 (degrees)
theta2 = angle from frame x axis to coupler x axis in position 2 (degrees)
theta3 = angle from frame x axis to coupler x axis in position 3 (degrees)
Xn     = x coordinate of slider point relative to coupler coordinate system
Yn     = y coordinate of slider point relative to coupler coordinate system
```

The slider point coordinates are returned in the vector "values". The coordinates are stored in "Values" according to the following:

```
Values(1) = slider point x in position 1 relative to frame system
Values(2) = slider point y in position 1 relative to frame system
Values(3) = slider point x in position 2 relative to frame system
Values(4) = slider point y in position 2 relative to frame system
Values(5) = slider point x in position 3 relative to frame system
Values(6) = slider point y in position 3 relative to frame system
Values(7) = slope of line relative to frame system (radians)
Values(8) = distance from point 1 to point 3.
```

4.A.2.3 Rectangle Routine (*rectangle.m*)

The rectangle routine determines the coordinates of a series of points defining a rectangle. The number of points stored is 5, and the first and last point are the same. The initial statement in the program is

```
function [coords] = rectangle(length,height,x0,y0,theta,flag)
```

The input variables are:

```
length  = length of rectangle
height  = height of rectangle
x0      = x coordinate of center or bottom left corner of rectangle
y0      = y coordinate of center or bottom left corner of rectangle
theta   = rotation angle relative to the horizontal x axis (degrees)
flag    = flag indicating if the rotation and (x0, y0) are defined
          relative to the center (flag = 0) or to the bottom left hand
          corner of the rectangle.
```

The results are returned in the vector "values". The answers are returned in values according to the following:

```
coords(i,1) = x coordinates of ith rectangle point
coords(i,2) = y coordinates of ith rectangle point
```

4.A.2.4 Slider-Crank Routine (*sldcrks.m*)

This function analyzes a slider crank mechanism when the slider is the driving link. The input values are:

```
r1      = length of vector 1
r2      = length of vector 2 (crank)
r3      = length of vector 3 (coupler)
r4      = length of vector 4 (slider offset)
rd1     = slider velocity
rdd1    = slider acceleration
sigma   = +1 or -1.  Identifies assembly mode
theta1  = angle between slider velocity and frame x axis (Degrees).
flag    = analysis flag.  If flag = 1, only a position analysis is conducted.
          If flag = 2, both a position and velocity analysis is conducted.
          If flag = 3, a position, velocity, and acceleration analysis
                       is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of crank pin (point Q)
values (27:28) = x,y components of position of piston pin (point P)
values (29:30) = x,y components of velocity of crank pin (point Q)
values (31:32) = x,y components of velocity of piston pin (point P)
values (33:34) = x,y components of acceleration of crank pin (point Q)
values (35:36) = x,y components of acceleration of piston pin (point P)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

4.A.2.5 Frame Line Routine (*frameline.m*)

The frameline routine determines the coordinates of a series of points defining the frame line for a mechanism. The number of points stored is 3*ndash. The initial statement in the program is

```
function [coords] = frameline(length,x0,y0,ndash,theta,flag)
```

The input variables are:

```
length = length of frame line
x0     = x coordinate of left end of line
y0     = y coordinate of left end of line
ndash  = number of dashes at -45 degrees to base
theta  = rotation angle relative to the horizontal x axis (degrees)
flag   = slide flag.  If flag = 1, the dashes are drawn on the bottom
         of the line.  If flag = -1, they are drawn on the top.
```

The results are returned in the vector "coords". The answers are stored in values according to the following:

```
coords(i,1) = x coordinates of ith dash of frame line
coords(i,2) = y coordinates of ith dash of frame line
```

4.A.2.6 Sample Run using *rbg_slidercrank.m*

The results from a sample run using the slider-crank synthesis program are summarized in Table 4.A.2, and the graphical results are shown in Figs. 4.A.3 and 4.A.4. The positions specified are the same as those used in the previous example. Note that in this example, the mechanism changes branch. This can be verified by animating the mechanism in both modes.

4.A.3 Design Routines for Rigid Body Guidance using a Crank-Slider Linkage

This set of functions includes a main routine (*rbg_crankslider*) and thirteen function routines. Eleven of the function routines are *axisadjust.m, circle.m*, *bush.m, frameline.m, pole.m, ipole.m, cosline.m, circlepoint.m, centerpoint.m, sliderpoint.m,* and *rectangle.m*, which were described previously. The two new routines are identified in the following.

4.A.3.1 Assembly Mode Routine (*assemblymode_cs.m*)

The assembly mode routine determines the assembly mode for a crank-slider linkage designed using rigid-body guidance techniques by checking the mode in the first position. The initial statement in the program is

```
function mode = assemblymode_cs(r2,r3,bx,by,Q1,Q2)
```

The input variables are:

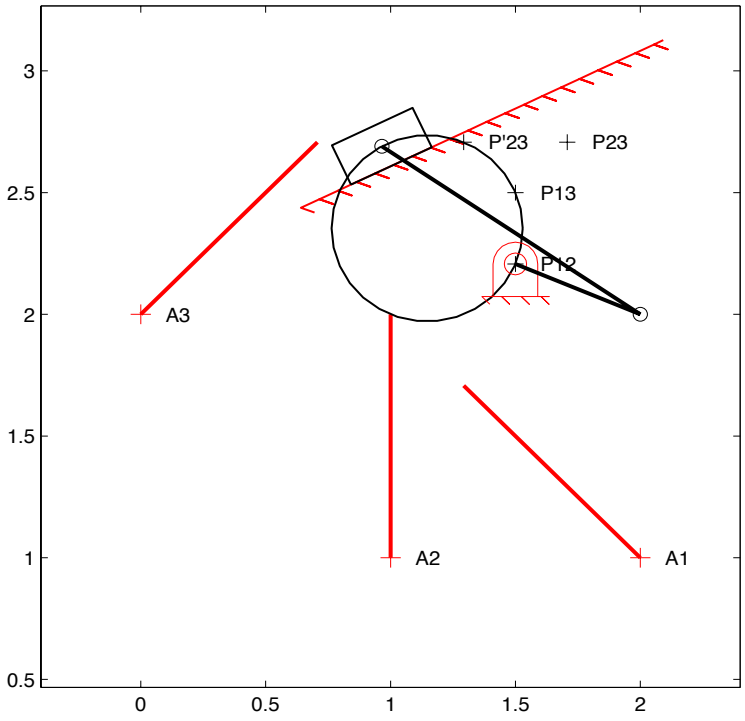


Fig. 4.A.3: Positions specified and basic slider-crank linkage designed

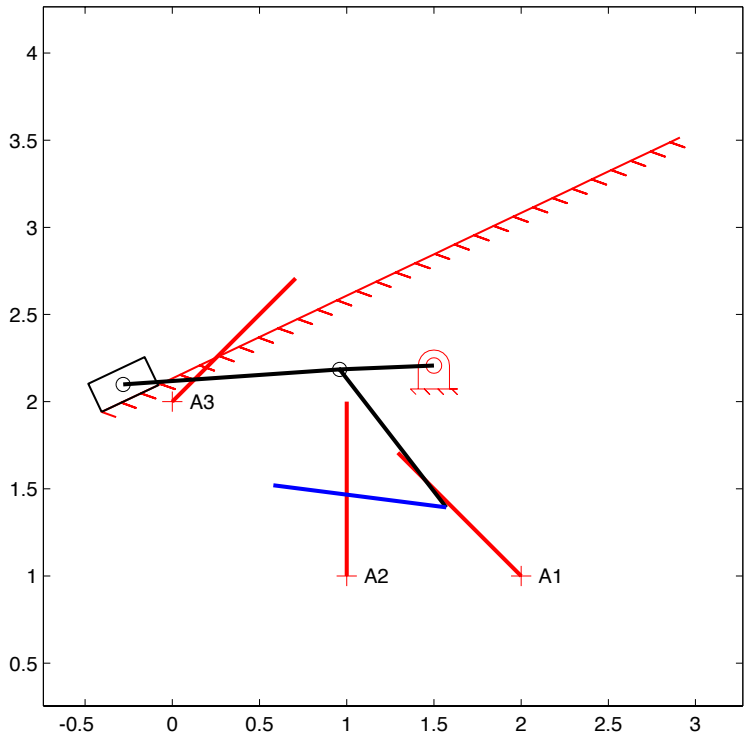


Fig. 4.A.4: Results after animation

Table 4.A.2: Input Data and Numerical Results from *rbg_slidercrank.m*

```
»rbg_slidercrank

         Slider-Crank Design for Rigid-Body Guidance
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (rgb_sldcrkio.dat): manual.dat
Enter value of ax1 [0]: 2
Enter value of ay1 [0]: 1
Enter value of theta1 (deg) [45]: 135
Enter value of ax2 [3]: 1
Enter value of ay2 [0]: 1
Enter value of theta2 (deg) [135]: 90
Enter value of ax3 [2]: 0
Enter value of ax3 [2]: 2
Enter value of theta3 (deg) [0]: 45

         Input Position Information
   Position     ax(i)      ay(i)     theta(i)
      1         2.000      1.000     135.000
      2         1.000      1.000      90.000
      3         0.000      2.000      45.000

Possible modes of inputting data:
  1 - mouse
  2 - keyboard
Designate mode of input [1] 2
```

```
Possible types of points to be input:
  1 - circle point
  2 - center point
Designate type of point to be input [1]: 1

 Input x component of circle point 2
 Input y component of circle point 2
reenter pivot point? y/n [n]: n
 Input x component of first slider point 0.96
 Input y component of first slider point 2.7
reenter slider point? y/n [n]: n
reenter pivot/slider points? y/n [n]: n

        Link Lengths and Slide Parameters
     r2       r3     offset   slidler ang(deg)
    0.54     1.24     0.67        25.39

 Center and Circle Point Coordinates in Position 1
         Ax      Ay      Astarx    Astary
        2.00     2.00     1.50     2.21

 Slider Point Coordinates in Position 1
             Bx       By
            0.97     2.69

Repeat animation? y/n [y]: n
Change assembly mode and repeat analysis? y/n [y]: n
```

```
r2, r3 = the link lengths numbered in the standard fashion (r2=driver,
         r3=coupler).
bx     = slider x coordinate relative to frame
by     = slider y coordinate relative to frame
Q1     = slide angle in degrees
Q2     = driver angle in degrees
```

The returned value is

```
mode = the assembly mode (+1 or -1)
```

4.A.3.2 Crank-Slider Routine (*sldcrkc.m*)

This function analyzes a slider-crank mechanism when the crank is the driving link. The input values are:

```
r2      = length of vector 2 (crank)
r3      = length of vector 3 (coupler)
r4      = length of vector 4 (slider offset)
theta2  = crank angle (degrees)
td2     = crank angular velocity (rad/sec)
tdd2    = crank angular acceleration (rad/sec^2)
sigma   = +1 or -1.  Identifies assembly mode
theta1  = angle between slider velocity and frame x axis (Degrees).
flag    = analysis flag.  If flag = 1, only a position analysis is conducted.
          If flag = 2, both a position and velocity analysis are conducted.
          If flag = 3, a position, velocity, and acceleration analysis
                       are conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:4)   = vector lengths
values (5:8)   = vector angles (degrees)
values (9:12)  = derivatives of vector lengths
values (13:16) = derivatives of vector angles (rad/sec)
values (17:20) = second derivatives of vector lengths
values (21:24) = second derivatives of vector angles (rad/sec^2)
values (25:26) = x,y components of position of crank pin (point Q)
values (27:28) = x,y components of position of piston pin (point P)
values (29:30) = x,y components of velocity of crank pin (point Q)
values (31:32) = x,y components of velocity of piston pin (point P)
values (33:34) = x,y components of acceleration of crank pin (point Q)
values (35:36) = x,y components of acceleration of piston pin (point P)
values (37)    = assembly flag.  If values(37) = 0, mechanism cannot
                 assembled.
```

4.A.3.3 Sample Run using *rbg_crankslider.m*

The results from a sample run using the crank-slider synthesis program are summarized in Table 4.A.3, and the graphical results are shown in Figs. 4.A.5 and 4.A.6. The positions specified are the same as those used in the previous examples. Note that in this example, the mechanism again changes branch. This can be verified by animating the mechanism in both modes.

Table 4.A.3: Input Data and Numerical Results from *rbg_crank_slider.m*

```
          Crank-Slider Design for Rigid-Body Guidance
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (rgb_crksldio.dat): manual.dat
Enter value of ax1 [0]: 2
Enter value of ay1 [0]: 1
Enter value of theta1 (deg) [45]: 135
Enter value of ax2 [3]: 1
Enter value of ay2 [0]: 1
Enter value of theta2 [135]: 90
Enter value of ax3 [2]: 0
Enter value of ax3 [2]: 2
Enter value of theta3 (deg) [0]: 45

          Input Position Information
   Position     ax(i)      ay(i)     theta(i)
      1         2.000      1.000     135.000
      2         1.000      1.000      90.000
      3         0.000      2.000      45.000

Possible modes of inputting data:
  1 - mouse
  2 - keyboard
Designate mode of input [1] 2
 Input x component of first slider point [1.26] 1.5
 Input y component of first slider point [1.99] 1.2
reenter slider point? y/n [n]: n

Possible types of points to be input:
  1 - circle point
  2 - center point
Designate type of point to be input [1]: 1

 Input x component of circle point [ax(1)] 1.9
```

```
 Input y component of fcircle point [ay(1)] 2.6
reenter pivot point? y/n [n]: n
reenter pivot/slider points? y/n [n]: n

        Link Lengths and Slide Parameters
     r2        r3      offset    slidler ang(deg)
    0.30      0.89      0.62        109.77


 Center and Circle Point Coordinates in Position 1
         Ax       Ay       Astarx     Astary
        1.90     2.60      1.79      2.32

 Slider Point Coordinates in Position 1
             Bx        By
           1.26      1.99

Repeat animation? y/n [y]: n
Change assembly mode and repeat analysis? y/n [y]: n
```

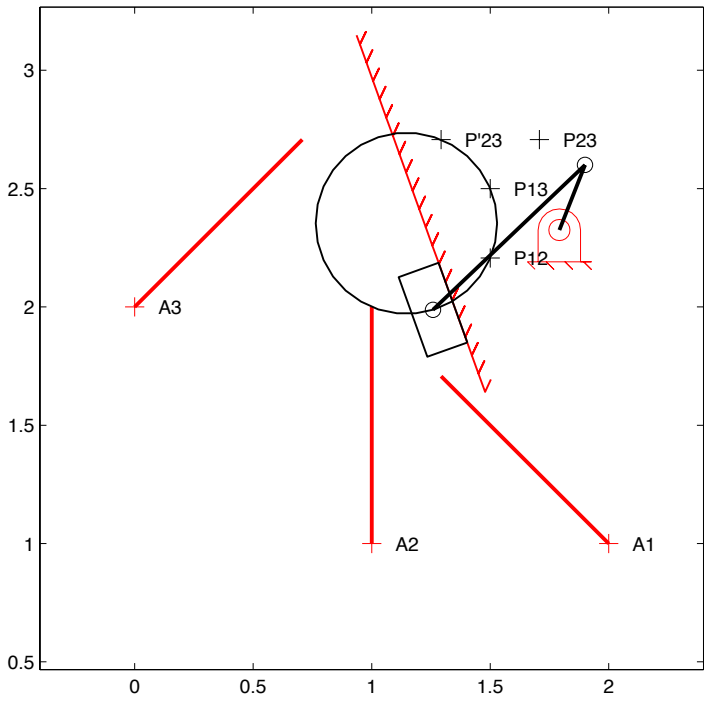


Fig. 4.A.5: Positions specified and basic crank-slider linkage designed

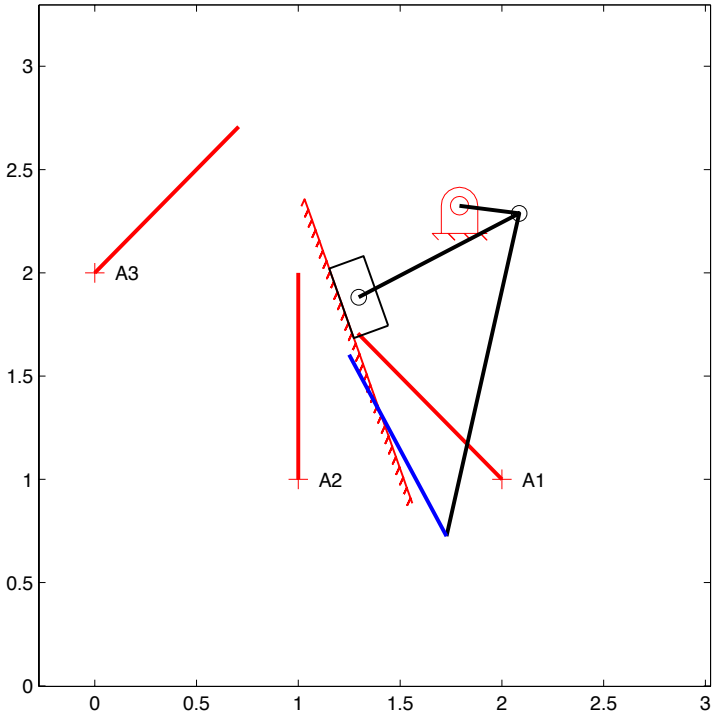


Fig. 4.A.6: Results after animation

4.A.4 Design Routines for Rigid Body Guidance using an Elliptic Trammel Linkage

This set of functions includes a main routine (*rbg_el_trammel.m*) and nine function routines, all of which have been defined previously. The function routines are: *axisadjust.m, circle.m*, *frameline.m, pole.m, ipole.m, cosline.m, sliderpoint.m, prrpc.m*, and *rectangle.m*.

4.A.4.1 Sample Run using rbg_el_trammel.m

The results from a sample run using the elliptic-trammel synthesis program are summarized in Table 4.A.4, and the graphical results are shown in Figs. 4.A.7 and 4.A.8.

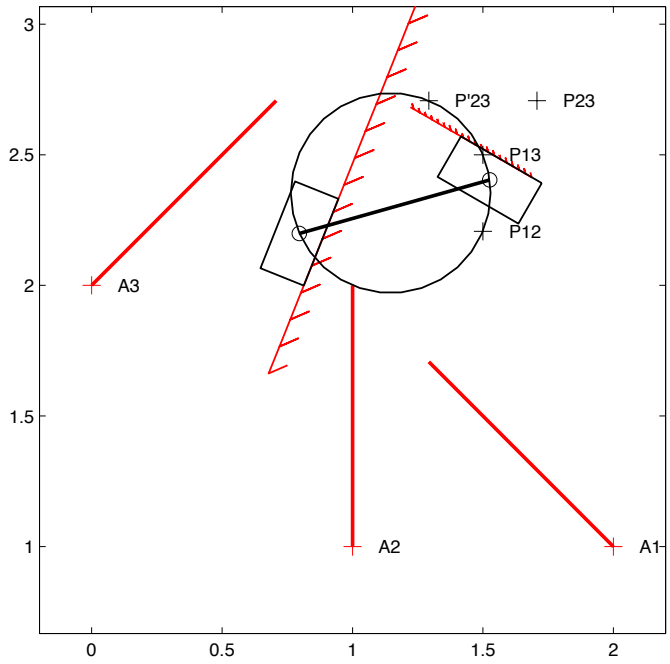


Fig. 4.A.7: Positions specified and basic elliptic trammel linkage designed

Table 4.A.4: Input Data and Numerical Results from *rbg_el_trammel.m*

```
        Elliptic Trammel Design for Rigid-Body Guidance
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (rbg_eltrammelio.dat): manual.dat

Enter value of ax1: 2
Enter value of ay1: 1
Enter value of theta1: 135
Enter value of ax2: 1
Enter value of ay2: 1
Enter value of theta2: 90
Enter value of ax3: 0
Enter value of ax3: 2
Enter value of theta3: 45

        Input Position Information
   Position    ax(i)     ay(i)    theta(i)
      1        2.000     1.000    135.000
      2        1.000     1.000     90.000
      3        0.000     2.000     45.000


Possible modes of inputting data:
  1 - mouse
  2 - keyboard
Designate mode of input [1] 2
 Input x component of first slider point 1.5
 Input y component of first slider point 2.4
reenter first slider point? y/n [n]: n
```

```
 Input x component of second slider point 0.8
 Input y component of second slider point 2.2
reenter second slider point? y/n [n]: n
reenter pivot/slider points? y/n [n]: n

   Link Lengths and Slide Parameters
     r3       slide1 ang(deg)    slide2 ang(deg)
    0.76          149.99             68.20

 Slider Point Coordinates in Position 1
       Bx1       By1       Bx2       By2
      1.53      2.40      0.80      2.20

Repeat animation? y/n [y]: n
»
```

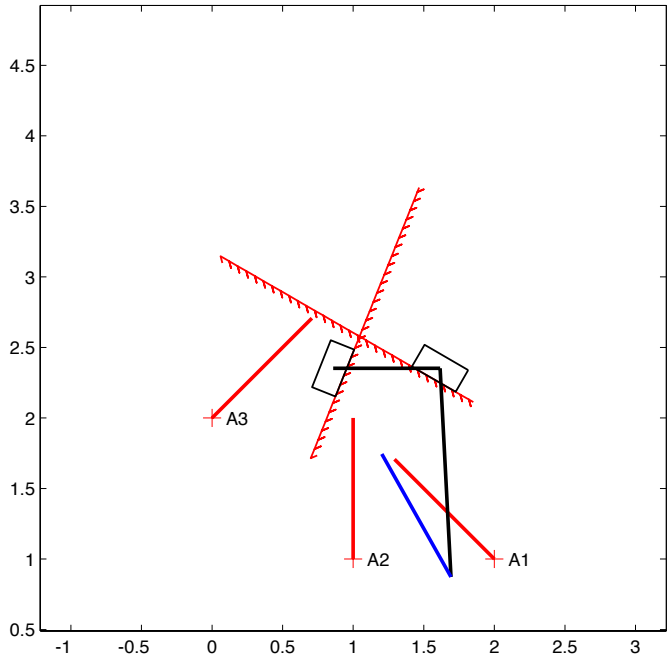


Fig. 4.A.8: Results after animation

4.B MATLAB Procedure for Function Generation

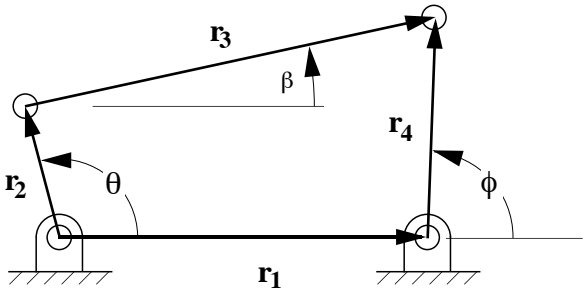


Fig. 4.B.1: Vector loop for function-generation mechanism

4.B.1 Overview

In this appendix, the computerization of the function generation problem introduced in Section 4.3 is presented. The procedure presented assumes that a four-bar linkage is to be synthesized and that θ is the input angle and ϕ is the output (see Fig. 4.B.1). The function of interest is y = f(x) where x is linearly related to θ, and y is linearly related to ϕ. The function is programmed in the m-file function *funct.m*, and this is the only routine which must be reprogrammed for different problems. It is also possible to input the values for θ and ϕ directly. In this case, the function *funct.m* can be a dummy routine.

The results are animated in the MATLAB program. For this, it is necessary to determine the type or classification for the linkage. The procedure for doing this is developed in the following.

4.B.2 Linkage Classification

To determine if the crank of the linkage can rotate 360˚, it is necessary to check Grashof's inequality. The linkage is a crank rocker if,

$$r_{max} + r_{min} < r_p + r_q$$

If the linkage is not a Grashof linkage, the limits of motion must be obtained if the linkage is to be analyzed for its full range of motion. Two cases must be considered.

4.B.2.1 Case when $r_2 = r_{min}$

This case can be obtained with the aid of Fig. 4.B.2.

Here,

$$\cos(\theta_{max,min}) = \pm\frac{\left[r_2^2 + r_1^2 - (r_3 + r_4)^2\right]}{2r_1r_2}$$

4.B.2.2 Case when $r_3 = r_{min}$ *or* $r_4 = r_{min}$

This case can be obtained with the aid of Fig. 4.B.3.

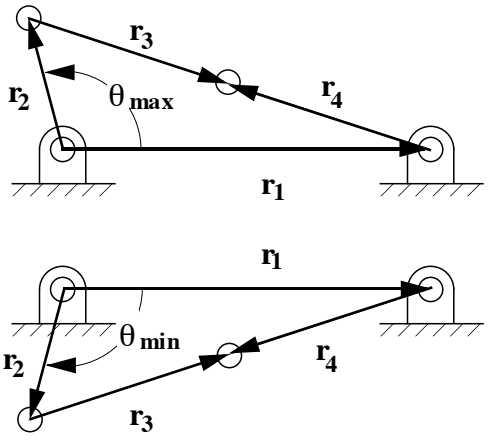


Fig. 4.B.2: Limits for input angle when r_2 is the shortest link

When the crank is above the line of centers, the angle limits are:

$$\cos(\theta_{max}) = \frac{\left[r_2^2 + r_1^2 - (r_3 + r_4)^2\right]}{2r_1r_2}$$

and

$$\cos(\theta_{min}) = \frac{\left[r_2^2 + r_1^2 - (r_4 - r_3)^2\right]}{2r_1r_2}$$

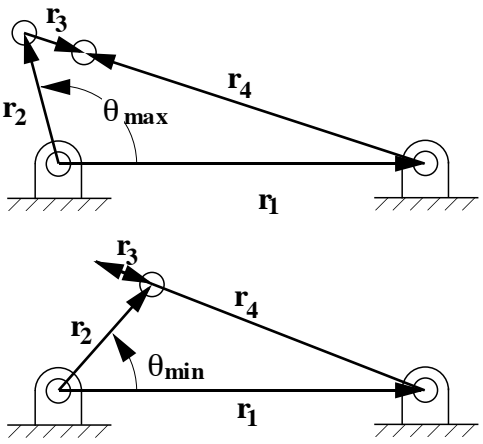


Fig. 4.B.3: Limits for input angle when r_3 is the shortest link

When the crank is below the line of centers, the angle limits are:

$$\cos(\theta_{max}) = -\frac{\left[r_2^2 + r_1^2 - (r_4 - r_3)^2\right]}{2r_1r_2}$$

and

$$\cos(\theta_{min}) = -\frac{\left[r_2^2 + r_1^2 - (r_3 + r_4)^2\right]}{2r_1r_2}$$

4.B.3 Function Routines

The MATLAB procedure involves the main routine (*functgen.m*) and eight specific m-file function routines. In addition, the routines *axisadjust.m, fourbar.m*, *bushing.m*, and *circle.m* are used to analyze the four-bar linkage and to determine the coordinates of the bushings and bushing pin so that the linkage can be animated. The names and purpose of the individual routines are given in the following:

4.B.3.1 Animation Routine (*ani.m*)

The animation routine animates the four-bar linkage used for function generation. The initial statement in the program is

```
function ani(r,Bx,By,Cx,Cy,Bx2,By2,Cx2,Cy2,ipos,phil,thetal)
```

The input variables are:

```
r       = vector of link lengths [r1 r2 r3 r4]
Bx, By  = x, y coordinates of the crank pin
Cx, Cy  = x, y coordinates of the rocker pin
```

```
Bx2, By2 = x,y coordinates of the crank pointer
Cx2, Cy2 = x,y coordinates of the rocker pointer
ipos    = number of animation positions
phil    = two component vector giving maximum and minimum limits for output
          angle ϕ.
thetal  = two component vector giving maximum and minimum limits for input
          angle θ.
```

4.B.3.2 Calculation Routine (*animat.m*)

The function *animat.m* calculates the output angle corresponding to the input angles at equal intervals. It also computes the coordinates for the linkage points used in animation. The initial statement in the program is

```
function [Q22,Q44,Bx,Cx,By,Cy,Bx2,Cx2,By2,Cy2] = animat (thetal, phil, r, mode,
r20,r40,ipos)
```

The input variables are:

```
phil    = two component vector giving maximum and minimum limits for rocker
          angle ϕ.
thetal  = two component vector giving maximum and minimum limits for crank
          angle θ.
r1=frame length (cm)
r2=crank length (cm)
r3=coupler length (cm)
r4=rocker length (cm)
r=[r1 r2 r3 r4]
mode=assembly mode
r20=length and direction of the crank
r40=length and direction of the rocker
ipos    = number of animation positions
```

The returned variables are:

```
Q22     = vector of crank angles - accounting for link direction
Q44     = vector of rocker angles - accounting for link direction
Bx,By   = x,y coordinates of the crank pin
Cx,Cy   = x,y coordinates of the rocker pin
Bx2,By2 = x,y coordinates of the crank pointer
Cx2,Cy2 = x,y coordinates of the rocker pointer
```

4.B.3.3 Precision point calculation routine (*chebychev.m*)

The function *chebychev.m* determines three x values corresponding to Chebychev spacing in the given range. The initial statement in the program is

```
function x=chebychev(xl)
```

The input variables are:

```
xl = two component vector giving limiting values of x
```

The returned values are:

```
x = three component vector of three precision points
```

4.B.3.4 Error routine (*errors.m*)

The function *errors.m* returns the error between the desired and generated values for y at each of the positions established for the linkage. The initial statement in the program is:

```
function errory=errors(yget,xl)
```

The input variables are:

```
yget = actual y values from linkage
xl   = limiting x values
```

The returned values are:

```
errory = relative error between desired and actual angular relationship
```

4.B.3.5 Function routine (*funct.m*)

The function *funct.m* returns a value for y given the value for x. The initial statement in the program is:

```
function y=funct(x)
```

The input variables are:

```
x=x value
```

The returned value is:

```
y = y value corresponding to the given x value.
```

An example routine for *funct.m* to compute y = log(x) is

```
function y=funct(x)

% Function file that contains the expression for the desired function.

% Variables
%   y=y value corresponding to the given x value
%   x=x value

% Program
y=log(x);
```

4.B.3.6 Link routine (*links.m*)

This function returns the values for the link lengths and linkage mode given the precision point values. The initial statement in the program is:

```
function [r, R, ra, mode, r20, r40, thetal] = links(theta, phi, lknown,
          rknown, thetalim)
```

The input variables are:

```
theta    = three component vector of crank angles
phi      = three component vector of rocker angles
lknown   = link with the known length
rknown   = known link length (cm)
thetalim = limiting crank angle (rad)
```

The returned values are:

```
r1     = frame length (cm)
r2     = crank length (cm)
r3     = coupler length (cm)
r4     = rocker length (cm)
r      = scaled link lengths [r1 r2 r3 r4]
R      = scaled link lengths with signs preserved (cm)
ra     = unscaled link lengths with signs preserved (cm)
r20    = length and direction of the crank
r40    = length and direction of the rocker
thetal = crank angle limits (deg)
```

4.B.3.7 Output value routine (*phi2y.m*)

This function returns a value for y given the value for ϕ. The initial statement in the program is:

```
function y=phi2y(phid,yl,phil)
```

The input variables are:

```
phid  = rocker angle at animation positions
yl    = limiting y values
phil  = limiting rocker angles
```

The returned values are:

```
y = y values corresponding to values of the rocker angles input
```

4.B.3.8 Output value routine (*precout.m*)

This function returns the values of ϕ and θ at the precision points given x and y. The initial statement in the program is:

```
function [theta,thi,yl] = precout(x,xl,phetal,phil,ansfun,ys, yf, yinput)
```

The input variables are:

```
x      = x values at the precision points
xl     = limiting x values
phetal = limiting crank angles (rad)
phil   = limiting rocker angles (rad)
ansfun = flag indicating if y is to be computed
ys     = lower limit for y
yf     = upper limit for y
yinput = y values at the precision points
```

The returned values are:

```
theta = crank angles at the precision points (rad)
```

```
thi   = rocker angles at the precision points (rad)
yl    = limiting y values, yl = [ys, yf]
```

The procedure solves the function generation problem when the function routine (*funct.m*) is supplied and we want to synthesize y = f(x). The precision points for x can be input directly or Chebychev spacing can be used.

Also, the precision points for both x and y can be input directly. This would be the case when we have only three points which need to be matched, i.e., no mathematical function is involved.

4.B.4 Sample results

As the first example, the problem in Example 4.3 has been solved using the procedure outlined. The input and output to the program *functgen.m* are summarized in Table 4.B.1 and the graphical results are given in Figs. 4.B.4 and 4.B.5. The numerical results compare well with the analytical results.

In the second example, the precision points are input directly. In this example, we specify the precision points such that effectively we specify ϕ as a function of θ. To do this, we make x = θ and y = ϕ. This can be done by setting the upper and lower limits of x and θ to be the same and setting the upper and lower limits of y and ϕ to be the same. The input and output for this case are summarized in Table 4.B.2 and the graphical results are given in Fig. 4.B.6. The function and error curves are not plotted because only three points are involved.

Table 4.B.1: Input and output corresponding to Example 5.3

```
                Function Generation Program

      Function Generation Program

Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (functgenio.dat): manual.dat
Enter number of cycles (rev) [3]: 2
Enter link number corresponding to known length [1]:  1
Enter known link length [2]: 2
Enter initial value for theta (deg) [45]: 45
Enter final value for theta (deg) [105]: 105
Enter initial rocker angle (deg) [0]: 0
Enter final rocker angle (deg) [60]: 60
Enter lower limit for x [1]: 1
Enter upper limit for x [2]: 2
Use Chebychev spacing for precision points? y/n [y]: y
Input y values directly? y/n [n]: n

    thetas     thetaf       phis       phif
    45.000    105.000      0.000     60.000

       xs         xf         ys         yf
     1.000      2.000      0.000      0.693

         x          y          theta        phi
     1.06699      0.06484      0.85555      0.09796
     1.50000      0.40547      1.30900      0.61257
```

```
     1.93301       0.65908       1.76245       0.99573

Repeat animation? y/n [y]: n

     Unscaled values for link lengths
       R1          R2          R3          R4
     1.000       0.795       2.030       1.508

       Scaled values for link lengths
       r1          r2          r3          r4
     2.000       1.591       4.061       3.016

Repeat animation? y/n [y]:
```

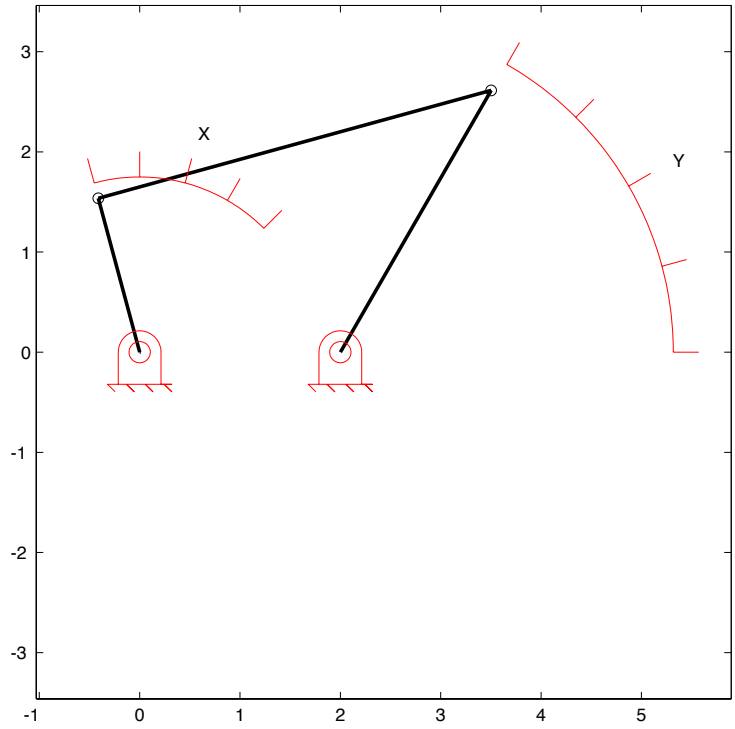


Fig. 4.B.4: Animation of linkage through ranges specified.

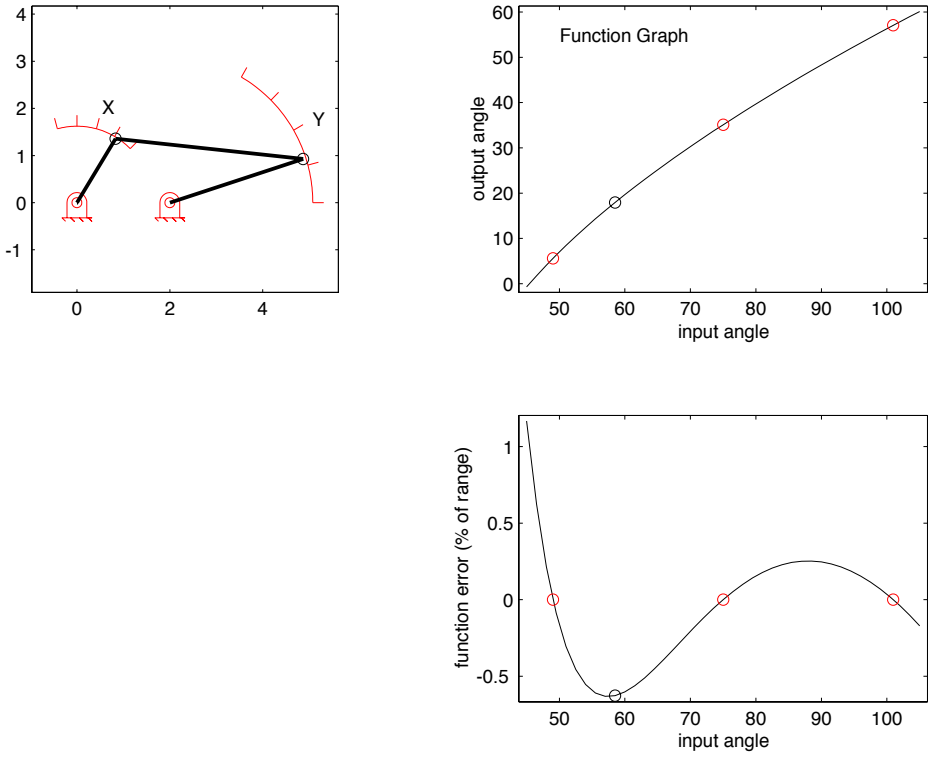


Fig. 4.B.5: Error curve for linkage.

Table 4.B.2: Input and output corresponding to second example (Direct input of precision points)

```
      Function Generation Program

Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (functgenio.dat): manual.dat
Enter number of cycles (rev) [3]:
Enter link number corresponding to known length [1]:  1
Enter known link length [2]: 1
Enter initial value for theta (deg) [45]: 330
Enter final value for theta (deg) [105]: 280
Enter initial rocker angle (deg) [0]: 80
Enter final rocker angle (deg) [60]: 140
Enter lower limit for x [1]: 330
Enter upper limit for x [2]: 280
Use Chebychev spacing for precision points? y/n [y]: n
Input y values directly? y/n [n]: y
Enter first x precision point: 330
Enter second x precision point: 310
Enter third x precision point: 280
Enter lower limit for y: 80
Enter upper limit for y: 140
Enter first y precision point: 80
Enter second y precision point: 110
Enter third y precision point: 140

    thetas     thetaf       phis       phif
```

```
   330.000     280.000      80.000     140.000

       xs          xf          ys          yf
   330.000     280.000      80.000     140.000

         x           y          theta         phi
   330.00000    80.00000      5.75959      1.39626
   310.00000   110.00000      5.41052      1.91986
   280.00000   140.00000      4.88692      2.44346

Repeat animation? y/n [y]: n

     Unscaled values for link lengths
       R1          R2          R3          R4
     1.000       0.461       0.940       0.429

       Scaled values for link lengths
       r1          r2          r3          r4
     1.000       0.461       0.940       0.429
```

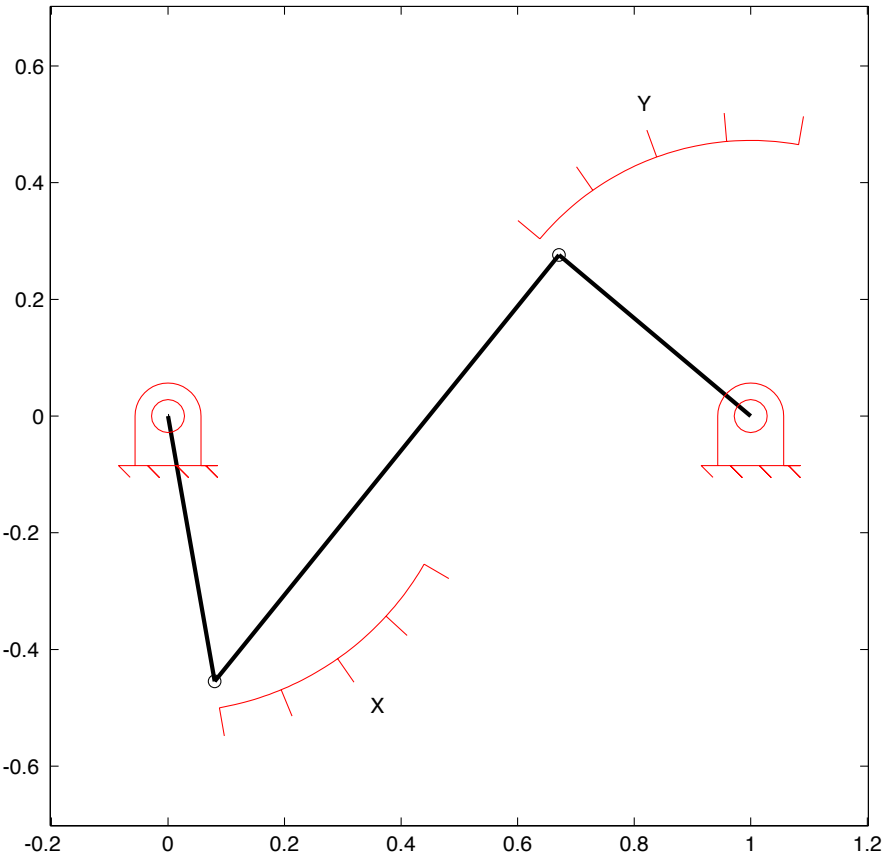


Fig. 4.B.6: Animation of linkage through ranges specified in second example.

4.C MATLAB Procedure for Crank Rocker Design

4.C.1 Overview

In this appendix, the computerization of the crank rocker design problem (Fig. 4.C.1) introduced in Section 4.4.3.2 is presented in m-files for MATLAB. The procedure presented assumes that a four-bar linkage is to be synthesized for a given output oscillation angle (θ) and time ratio (Q). From the time ratio, the angle α can be computed using Eq. (5.56). Given θ, α, the locus for B_2 can be determined. Once the user selects a given value for B_1, the normalized link lengths can be computed using the equations in Section 4.4.3.2. Given one of the link lengths, the proper linkage dimensions can be determined.

The m.file used for the calculations is called *crank_rocker_design.m*. The routines uses four other MATLAB routines. These are: *axisadjust.m, circle.m*, *bushing.m*, and *fourbar.m*. All of these have already been discussed. The results from a sample run are given following the source code.

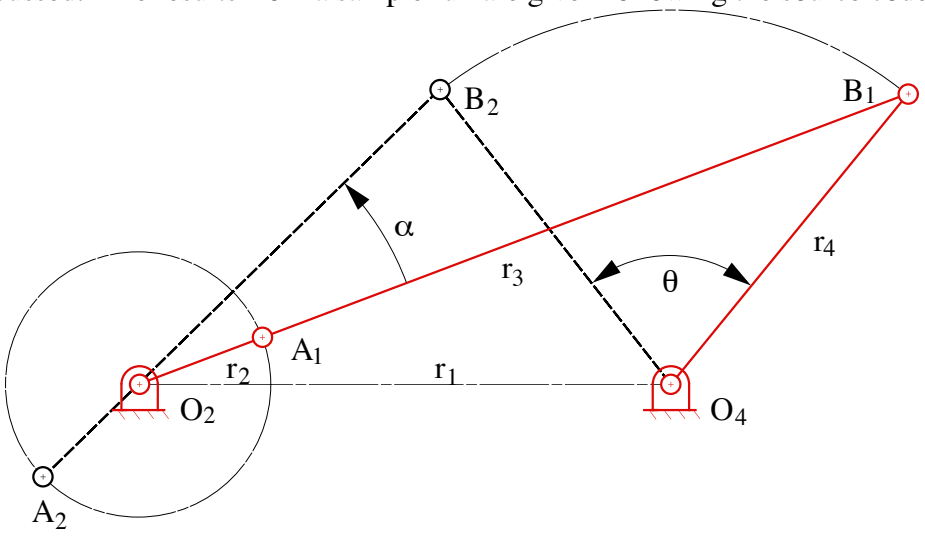


Fig. 4.C.1: Extreme Positions For Crank Rocker Mechanism

4.C.2 Results from Sample Run of Crank-Rocker Synthesis

In the sample run, the objective is to design a crank-rocker mechanism for a rocker angle of 80 degrees and a time ratio of 1.3. The input data is given in Table 4.C.1 and the graphical results are given in Figs. 4.C.2, 4.C.3, and 4.C.4.

Table 4.C.1: Input Data and Numerical Results from *crank_rocker_design.m*

```
            Crank-Rocker Synthesis Program

Enter link number corresponding to known length [1]:  1
Enter known link length [2]: 2
Enter rocker oscillation angle (deg) [80]: 80
Enter time ratio (Q) or alpha (a)? [a]:  Q
Enter time ratio for mechanism [1.32259065]: 1.2
Reenter input variables? y/n [n]: n

    theta (deg)    Q     alpha (deg)
    80.000       1.200      16.364
Possible modes of inputting data:
  1 - mouse
  2 - keyboard
```

```
Designate mode of input [1] 2
 Input beta (in deg) to locate B2 120
Reenter point? y/n [n]: n

     Scaled values for link lengths
       r1         r2         r3         r4
     1.000      0.486      0.934      0.777

       Actual values for link lengths
       R1         R2         R3         R4
     2.000      0.972      1.868      1.554
Repeat animation? y/n [y]: n

Select another design? y/n [y]: n
```

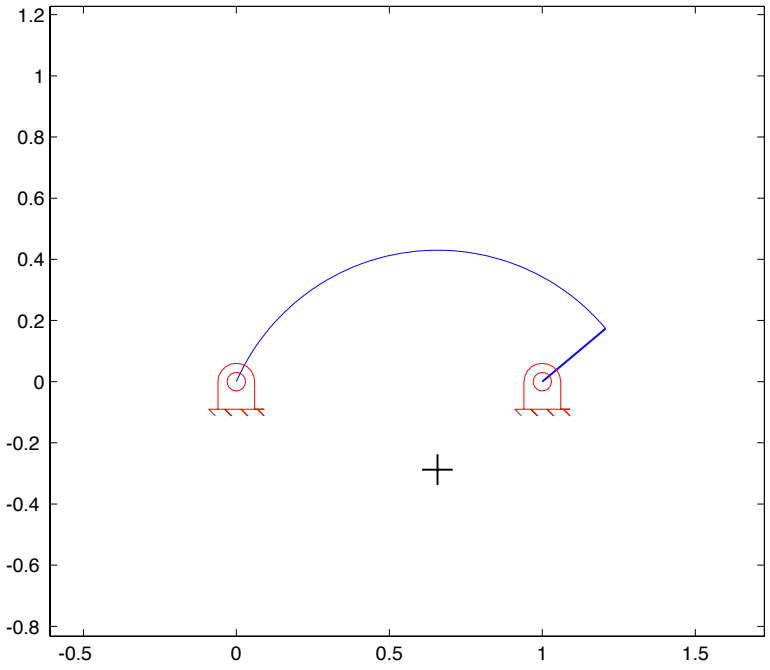


Fig. 4.C.2: Design showing circle for B2.

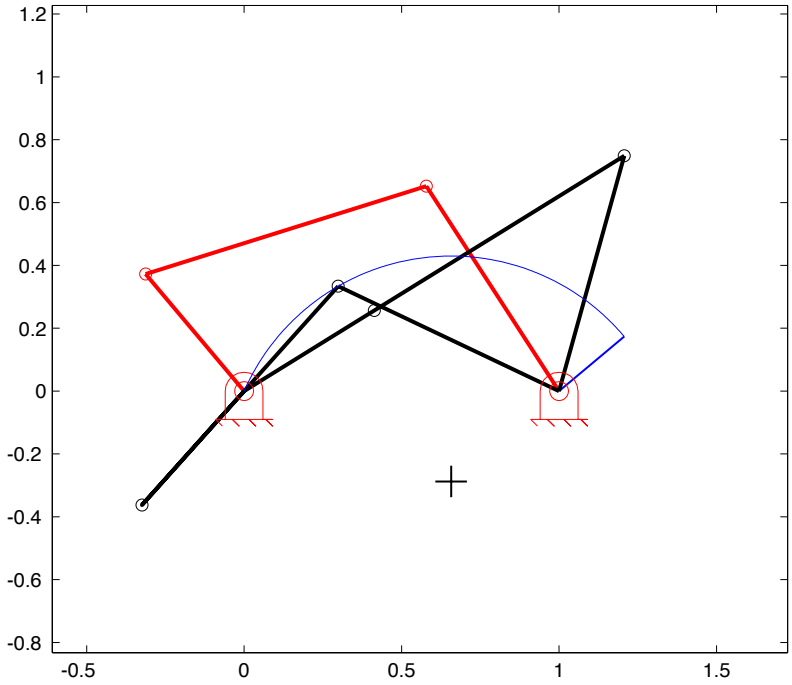


Fig. 4.C.3: Procedure with B2 selected

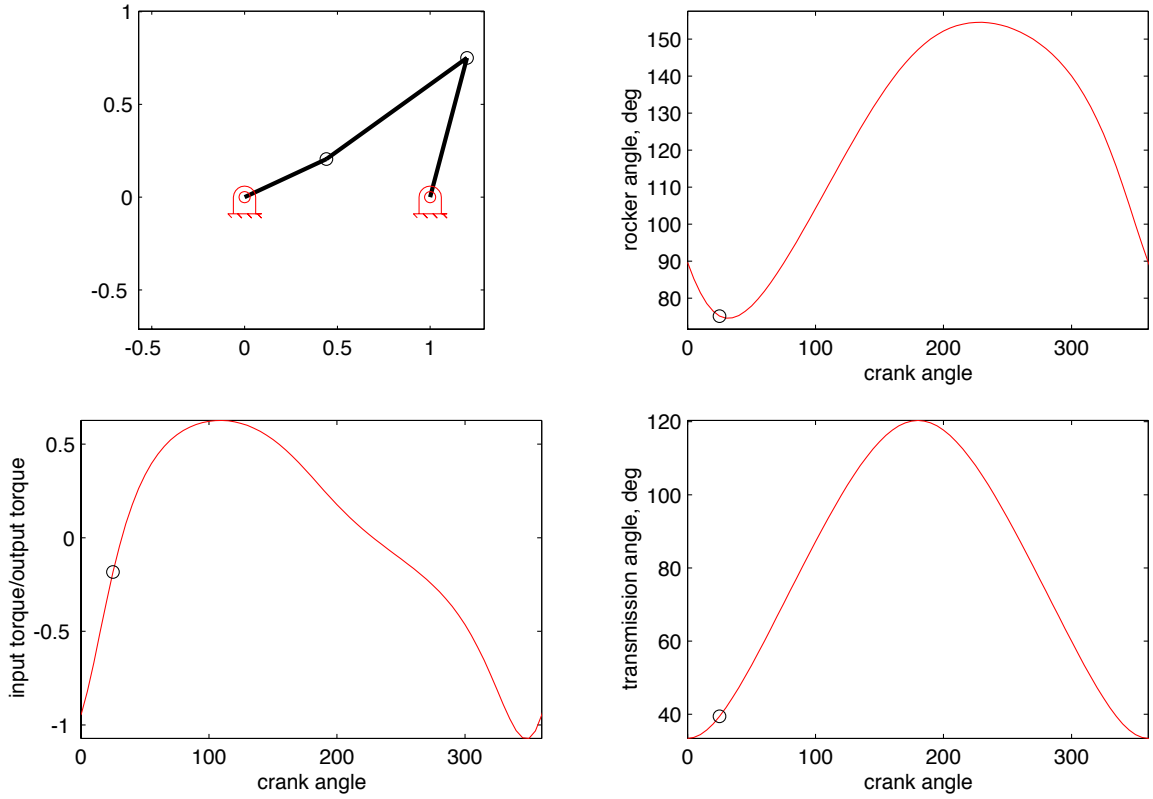


Fig. 4.C.4: Analysis of final design.

4.D MATLAB Procedure for Generating Coupler Curves

4.D.1 Overview

The appendix describes two m-files which can be used for generating coupler curves. One routine (*hr_crankrocker.m*) generates coupler curves for crank-rocker mechanisms, and the second (*hr_slidercrank.m*) generates coupler curves for a slider-crank mechanism. Both routines are modeled after the coupler curve atlas developed by Hrones and Nelson and described in Chapter 5. In both routines, a uniform grid of coupler points are assumed for the coupler, and the user is asked to select one row of points to evaluate. The user specifies the number of rows and number of points in each row. The coupler grid is located symmetrically relative to the two pivots on the coupler. The coupler curves are plotted with dashed lines where each dash corresponds to 5° of crank rotation. Each routine is described separately in the following.

4.D.2 Crank-rocker Routine (*hr_crankrocker.m*)

In the crank-rocker program, the driver link is assumed to be of unit length ($r_2 = 1$) and the other link lengths are input. Also, the number of points in each row and number of columns are input. The grid of points then appear as shown in Fig. 4.D.1. The program then asks the user to select the row of points to be evaluated and the program then generates the coupler curves for that row. Next the user is asked to select the specific coupler curve for further evaluation. Each coupler point is defined by the radial distance from the crank moving pivot (A) and the angle from the coupler line (AB) as shown in Fig. 4.D.1. The selected curve is then displayed separately, and finally, four plots are displayed showing the linkage , rocker position and velocity as a function of the crank angle, and the magnitude of the velocity of the coupler point as a function of the crank angle.

The results of a sample analysis are given in the following section.

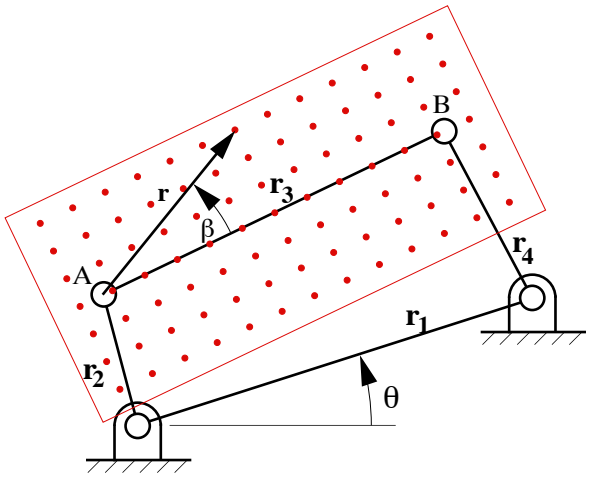


Fig. 4.D.1: Coupler point grid used in crank-rocker coupler-curve program.

4.D.2.1 Sample run using *hr_crankrocker.m*

In the following, a copy of the input screen is given in Table 4.D.1 and the plots are displayed in Figs. 4.D.2 - 4.D.5.

Table 4.D.1: Input and output corresponding to sample analysis

---

```
Crank-Rocker Coupler Curve Program
```

```
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (hrcrankrockerio.dat): manual.dat
Enter number of cycles (rev) [3]: 3
Enter the frame length [3.304]: 3
Enter the coupler length [2.913]: 3
Enter the rocker length [1.565]: 2
Enter the angular velocity of crank(rad/sec) [5]: 1
Enter total length of coupler grid [4*r3]: 5
Enter total height of coupler grid [2*r3]: 3
Enter number of coupler points in horizontal direction [20]: 8
Enter number of coupler points in vertical direction [10]: 5
Animation grid
Use mouse to indicate row to animate
I am working ...

repeat animation? y/n [y]: n
Use mouse to input point to animate

repeat animation? y/n [y]: n
Repeat animation? y/n [y]: n
Repeat animation for a new set of grid points? y/n [y]: n
```

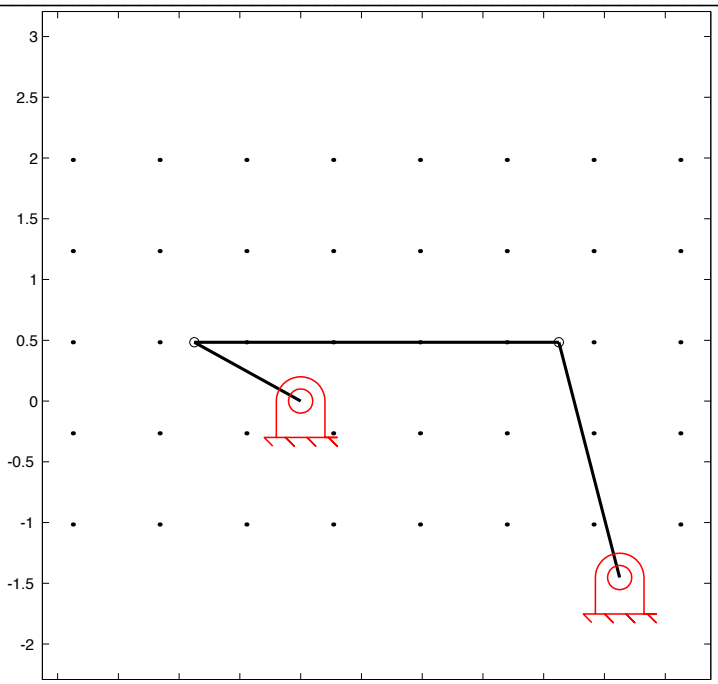


Fig. 4.D.2: Gridpoint plot for sample analysis.

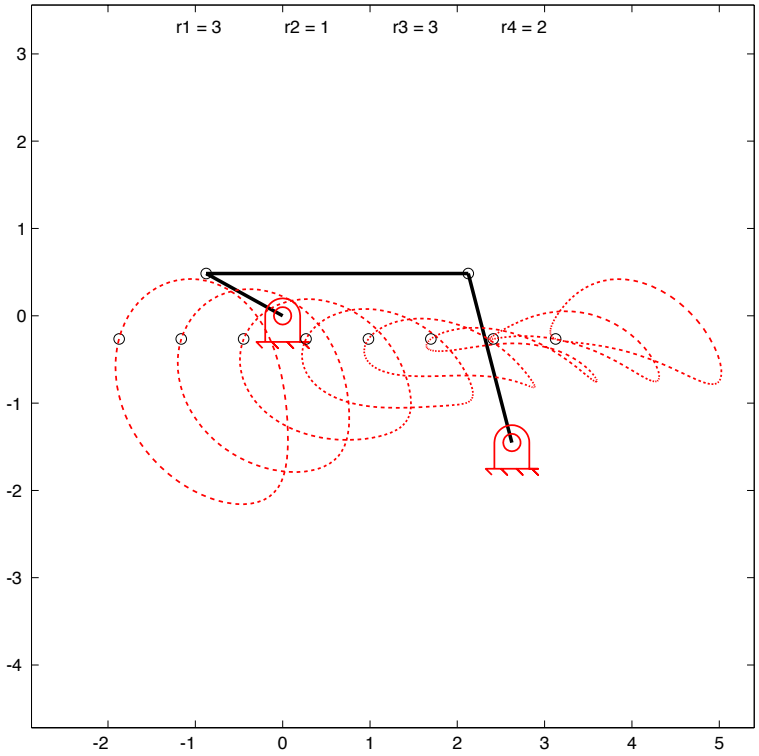


Fig. 4.D.3: Coupler curves for row selected.

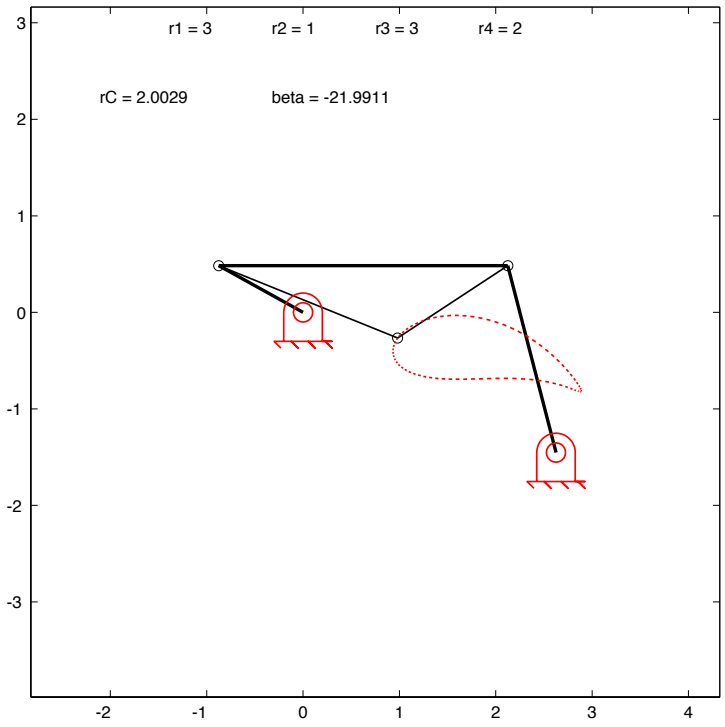


Fig. 4.D.4: Specific coupler curve selected for detailed analysis.

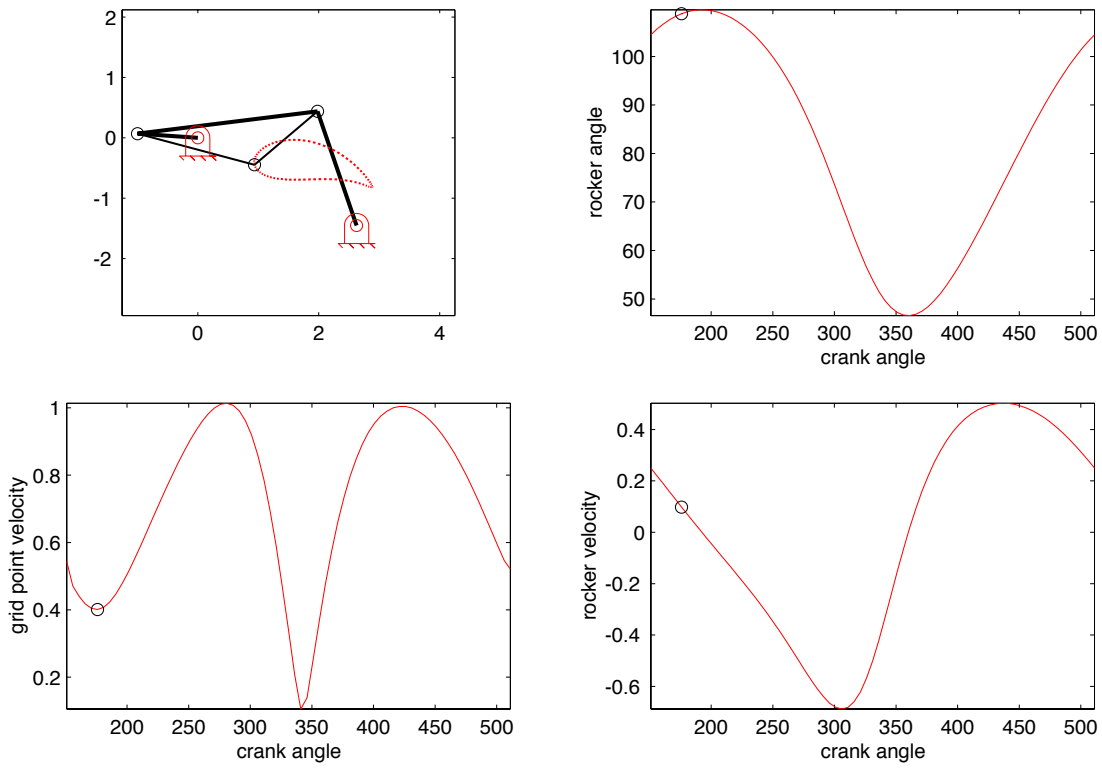


Fig. 4.D.5: Analysis of specific linkage selected.

4.D.2.2 MATLAB Routine for Crank-Rocker Coupler Curve Program

The MATLAB procedure involves the main routine (*hr_crankrocker.m*) and the routines *axisadjust.m, fourbar.m*, *bushing.m*, and *circle.m* and *rbody1.m* are used to analyze the four-bar linkage and to determine the coordinates of the bushings and bushing pin so that the linkage can be animated. These routines have been described previously, and the names and purpose of the individual routines are given in the following:

axisadjust.m: Function routine to adjust the axis limits so that the viewport and window in Matlab 5.0 is closer to that in Matlab 4.2 when the command "axis equal" is used

circle.m: This function returns the coordinates for a circle given the center location and radius

bushing.m: This function returns the coordinates for drawing a bushing or frame pivot.

fourbar.m: This function analyzes a four-bar linkage for position, velocity, and acceleration.

rbody1.m: This function analyzes the third point on a rigid body when the kinematic information for two other points are given.

4.D.3 Slider-Crank Routine (*hr_slidercrank.m*)

The slider-crank routine is similar to the crank-rocker routine. The lengths of the crank, coupler, and slider offset must be input along with the size of the coupler plate and the number of points in each row and number of columns. The grid of points appear as shown in Fig. 4.D.6. The program then asks the user to select the row of points to be evaluated and the program then generates the coupler curves for that row. Next the user is asked to select the specific coupler curve for further evaluation. Each coupler point is

defined by the radial distance from the crank moving pivot (A) and the angle from the coupler line (AB) as shown in Fig. 4.D.6. The selected curve is then displayed separately, and finally, four plots are displayed showing the linkage, rocker position and velocity as a function of the crank angle, and the magnitude of the velocity of the coupler point as a function of the crank angle.

The results of a sample analysis are given in the following section.

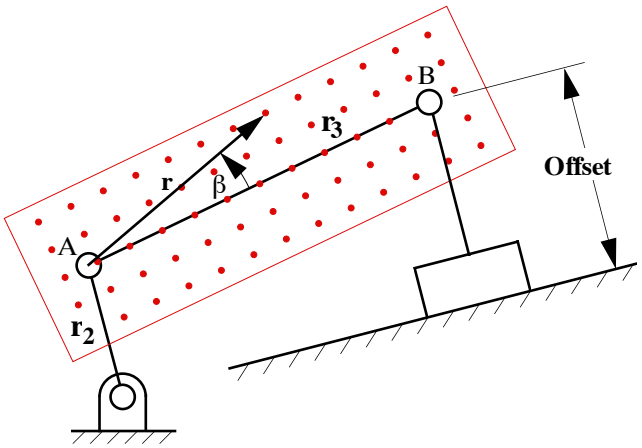


Fig. 4.D.6: Coupler point grid used in slider-crank coupler-curve program.

4.D.3.1 Sample run using *hr_slidercrank.m*

In the following, a copy of the input screen is given in Table 4.D.2 and the plots are displayed in Figs. 4.D.7 - 4.D.10.

Table 4.D.2: Input and output corresponding to sample analysis for slider-crank

```
        Slider-Crank Coupler Curve Program
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (hrslidercrankio.dat): manual.dat
Enter number of cycles (rev) [3]: 3
Enter slider-line offset [1]: 0.5
Enter the crank length [0.75]: 1
Enter the coupler length [2.75]: 3
Enter the angular velocity of crank(rad/sec) [5]: 1
Enter total length of coupler grid [4*r3]: 4
Enter total height of coupler grid [2*r2]: 4
Enter number of coupler points in horizontal direction [20]: 8
Enter number of coupler points in vertical direction [10]: 5
Use mouse to indicate row to animate

I am working ...

repeat animation? y/n [y]: n

Use mouse to input point to animate
repeat animation? y/n [y]: n
Repeat animation? y/n [y]:
```

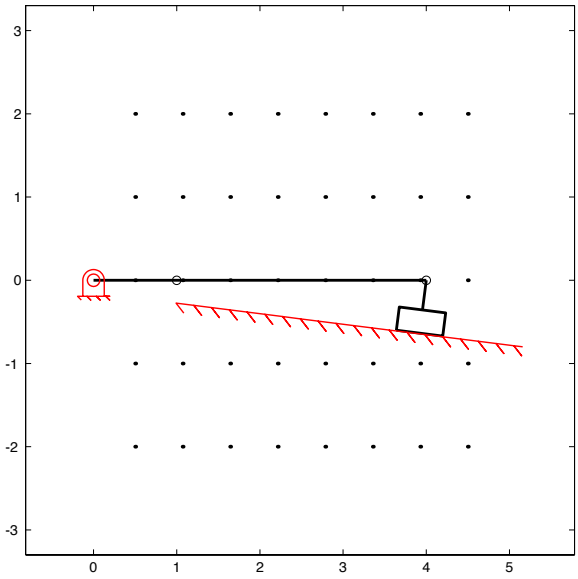


Fig. 4.D.7: Gridpoint plot for sample slider-crank analysis.

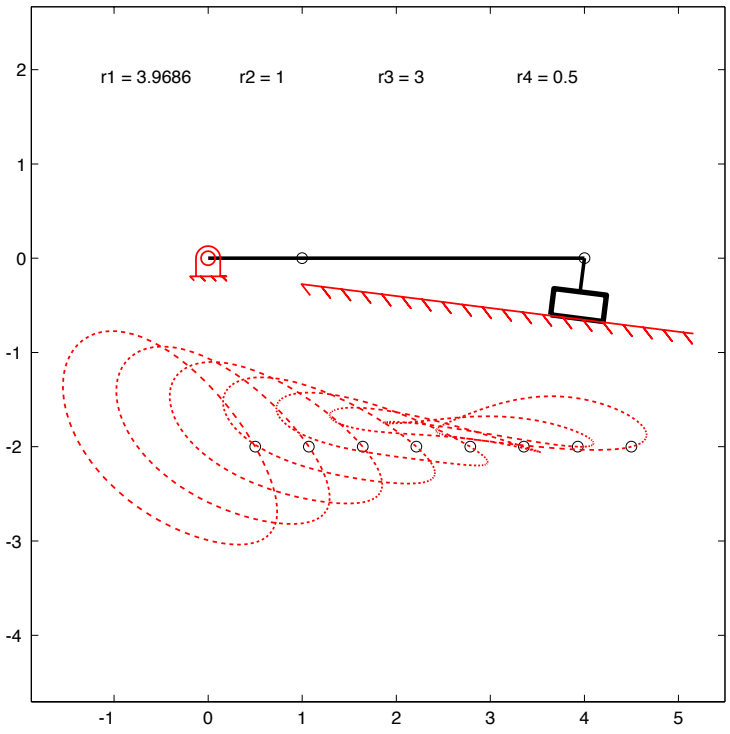


Fig. 4.D.8: Coupler curves for row selected.

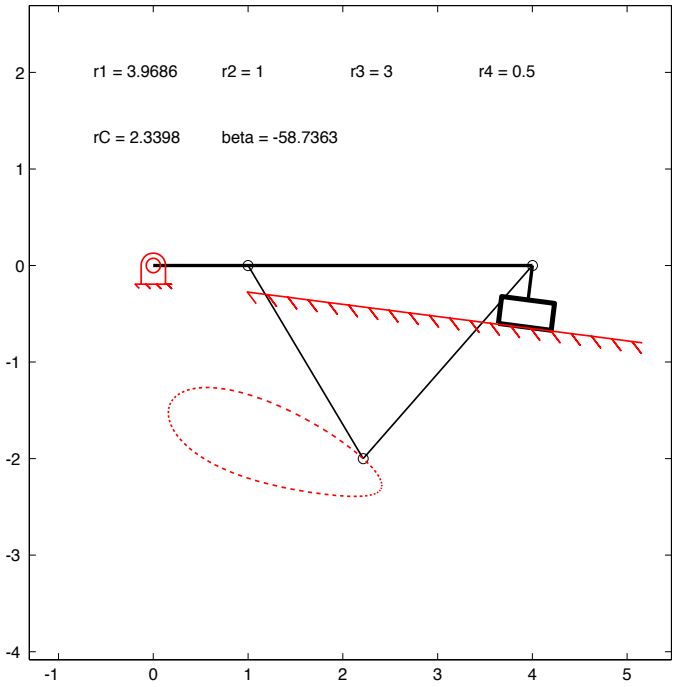


Fig. 4.D.9: Specific coupler curve selected for detailed analysis.

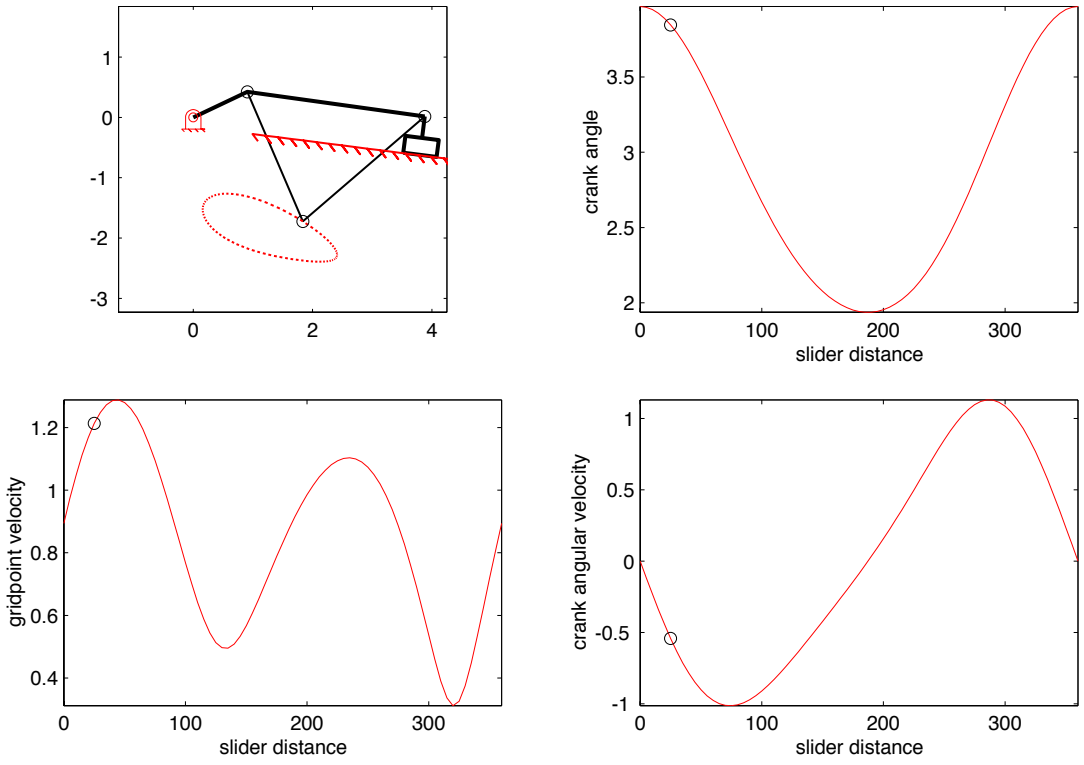


Fig. 4.D.10: Analysis of specific linkage selected.

4.D.3.2 MATLAB Routine for Slider-Crank Coupler Curve Program

The MATLAB procedure involves the main routine (*hr_slidercrank.m*) and the routines *axisadjust.m*, *bushing.m*, *circle.m*, *rectangle.m, frameline.m, slidcrkc.m,* and *rbody1.m* are used to analyze the four-bar linkage and to determine the coordinates of the bushings and bushing pin so that the linkage can be animated. These routines have been described previously, and the names and purpose of the individual routines are given in the following:

axisadjust.m: Function routine to adjust the axis limits so that the viewport and window in Matlab 5.0 is closer to that in Matlab 4.2 when the command "axis equal" is used

circle.m: This function returns the coordinates for a circle given the center location and radius

bushing.m: This function returns the coordinates for drawing a bushing or frame pivot.

rbody1.m: This function analyzes the third point on a rigid body when the kinematic information for two other points are given.

rectangle.m: This function returns the coordinates for a rectangle given the corner coordinates and the length and height.

frameline.m: This function returns the coordinates for drawing a frame line (hatched line).

slidcrkc.m: This function analyzes a slider-crank mechanism when the crank is driving.

4.E MATLAB Procedure for Crank and Dyad Analysis

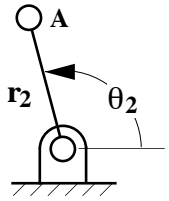


Fig. 4.E.1: Crank

4.E.1 Overview

In this appendix, equations for the analysis of a crank and dyad are programmed. With these two components, a four-bar linkage can be analyzed in addition to more complex mechanisms.

4.E.2 Crank Analysis

The crank is assumed to be connected to the frame by a revolute joint as shown in Fig. 4.E.1. The routine returns the values for the position, velocity, and acceleration of the endpoint A given the angular position, velocity, and acceleration of the link. The position, velocity, and acceleration equations are:

$$x_A = r_2 \cos\theta_2$$
$$y_A = r_2 \sin\theta_2 \text{ ,}$$

$$v_{xA} = -r_2\dot{\theta}_2 \sin\theta_2$$
$$v_{yA} = r_2\dot{\theta}_2 \cos\theta_2 \text{ ,}$$

and

$$a_{xA} = -r_2\ddot{\theta}_2 \sin\theta_2 - r_2\dot{\theta}_2^2 \cos\theta_2$$
$$a_{yA} = r_2\ddot{\theta}_2 \cos\theta_2 - r_2\dot{\theta}_2^2 \sin\theta_2$$

4.E.3 MATLAB Routine for Computing Crank Variables

This MATLAB file *crank.m* analyzes a crank AB when the angular position, velocity, and acceleration of the crank are given along with the linear position, velocity, and acceleration of point A. The position, velocity, and acceleration of B are to be found. The initial statement for the MATLAB m.file is

```
function [values] = crank(r2,theta2,td2,tdd2,flag)
```

The input variables are:

```
r2      = length of vector 2 (crank)
theta2  = crank angle (degrees)
td2     = crank angular velocity (rad/sec)
tdd2    = crank angular acceleration (rad/sec^2)
flag    = analysis flag.  If flag = 1, only a position analysis is
          conducted.  If flag = 2, both a position and velocity analysis is
          conducted.  If flag = 3, a position, velocity, and acceleration
          analysis is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values (1:2) = x,y components of position of crank pin
values (3:4) = x,y components of velocity of crank pin
values (5:6) = x,y components of acceleration of crank pin
```

4.E.4 Dyad Analysis

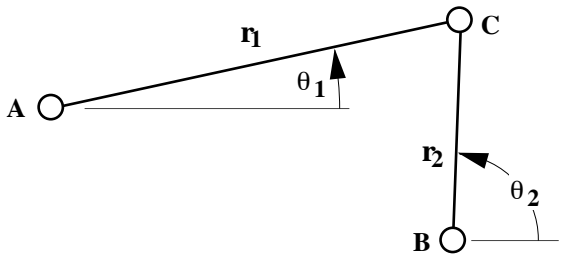


Fig. 4.E.2: Dyad

Referring to Fig. 4.E.2, it is assumed that the position, velocity, and acceleration of points A and B are known, and the position velocity, and acceleration of point C are to be found. The link lengths (r_1 and r_2) are also known along with the assembly mode. The procedure is programmed in the m-file function *dyad.m*.

The procedure solves the position equations given by:

$$\begin{aligned} x_A + r_1\cos\theta_1 &= x_B + r_2\cos\theta_2 \\ y_A + r_1\sin\theta_1 &= y_B + r_2\sin\theta_2 \end{aligned};$$

the velocity equations given by:

$$\begin{aligned} v_{xA} - r_1\dot{\theta}_1\sin\theta_1 &= v_{xB} - r_2\dot{\theta}_2\sin\theta_2 \\ v_{yA} + r_1\dot{\theta}_1\cos\theta_1 &= v_{yB} + r_2\dot{\theta}_2\cos\theta_2 \end{aligned};$$

and the acceleration equations given by:

$$a_{xA} - r_1\ddot{\theta}_1 \sin\theta_1 - r_1\dot{\theta}_1^2 \cos\theta_1 = a_{xB} - r_2\ddot{\theta}_2 \sin\theta_2 - r_2\dot{\theta}_2^2 \cos\theta_2$$
$$a_{yA} + r_1\ddot{\theta}_1 \cos\theta_1 - r_1\dot{\theta}_1^2 \sin\theta_1 = a_{yB} + r_2\ddot{\theta}_2 \cos\theta_2 - r_2\dot{\theta}_2^2 \sin\theta_2$$

4.E.5 MATLAB Routine for Solving Dyad Equations

This MATLAB file *dyad.m* analyzes a dyad (ABC in Fig. 4.E.2) when the position, velocity, and acceleration of A and B are given, and the position, velocity and acceleration of C are to be found The initial statement for the MATLAB m.file is

```
function [values] = dyad(r1, r2, xA, yA, xdA, ydA, xddA, yddA, xB, yB, xdB,
                    ydB, xddB, yddB, sigma,flag)
```

The input variables are:

```
r1        = length of first link
r2        = length of second link
xA        = x component of point A
yA        = y component of point A
xdA       = x component of velocity of point A
ydA       = y component of velocity of point A
xddA      = x component of aceleration of point A
yddA      = y component of aceleration of point A
xB        = x component of point B
yB        = y component of point B
xdB       = x component of velocity of point B
ydB       = y component of velocity of point B
xddB      = x component of aceleration of point B
yddB      = y component of aceleration of point B
sigma     = +1 or -1.  Identifies assembly mode
flag      = analysis flag.  If flag = 1, only a position analysis is
            conducted. If flag = 2, both a position and velocity analysis
            is conducted.  If flag = 3, a position, velocity, and
            acceleration analysis is conducted.
```

The results are returned in the vector "values". The answers are stored in values according to the following:

```
values(1:2)   = x,y coordinates of point C
values(3:4)   = angular position of links 3 and 4 (deg)
values(5:6)   = x,y coordinates of velocity of point C
values(7:8)   = angular velocity of links 3 and 4 (rad/sec)
values(9:10)  = x,y coordinates of acceleration of point C
values(11:12) = angular acceleration of links 3 and 4 (rad/sec^2)
values(13)    = assembly flag.  If values(13) = 0, mechanism cannot
                assembled.
```

4.F MATLAB Procedure for 6-Link Dwell Mechanism Analysis

4.F.1 Introduction

In this appendix, equations for the analysis of the 6-bar mechanism in Fig. 4.F.1 are programmed. This linkage is one of the Watt's type (Fig. 1.20). The analysis is conducted by treating the 6-link mechanism as an assembly of a 4-bar linkage, a rigid body (coupler) and a dyad. The three components are identified in Fig. 4.F.2. The equations and routines for these three components have been discussed earlier.

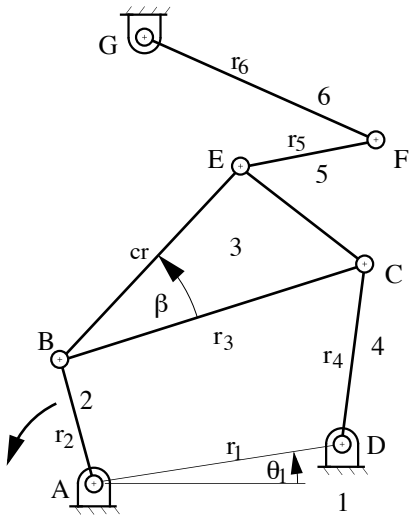


Fig. 4.F.1: 6-link Dwell Mechanism.

The MATLAB procedure involves the main routine (*sixbar.m)* and the six routines *axisadjust.m*, *fourbar.m*, *bushing.m*, *circle.m*, *rbody1.m*, *crank.m*, and *dyad.m*. These routines have been described previously, and the names and purpose of the individual routines are given in the following:

axisadjust.m: Function routine to adjust the axis limits so that the viewport and window in Matlab 5.0 are similar to those in Matlab 4.2 when the command "axis equal" is used

circle.m: This function returns the coordinates for a circle given the center location and radius

bushing.m: This function returns the coordinates for drawing a bushing or frame pivot.

fourbar.m: This function analyzes a four-bar linkage for position, velocity, and acceleration.

rbody1.m: This function analyzes the third point on a rigid body when the kinematic information for two other points are given.

crank.m: This function analyzes a simple crank for position, velocity, and acceleration.

dyad.m: This function analyzes the third point on a dyad when the kinematic information for the other two endpoints on the dyad is known.

After the basic data are input based on the nomenclature given in Fig. 4.F.1, the mechanism is analyzed and animated for a full cycle of motion. Next the linkage and three other plots are given. The plots show the rocker angle of the basic 4-bar linkage and the oscillation angle and angular velocity of link 6 as a function of the crank angle of the 4 bar linkage. The results of a sample analysis are given in the following section.

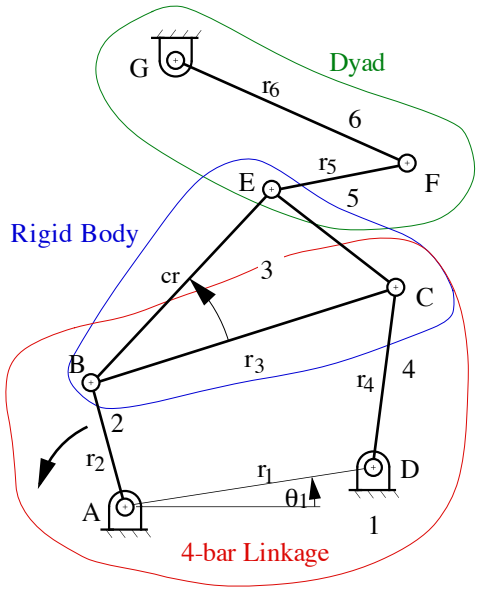


Fig. 4.F.2: 6-link mechanism made up of a four-bar linkage, a rigid body, and a dyad.

4.F.2 Sample Run Using *sixbar.m*

In the following, a copy of the input screen is given in Table 4.F.1 and the plots are displayed in Figs. 4.F.3 - 4.F.4.

Table 4.F.1: Input and output corresponding to sample analysis

```
        Sixbar Dwell Mechanism Analysis Program
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (sixbario.dat): manual.dat
Enter number of cycles (rev) [3]: 3
Enter the frame length (cm) [4]: 4
Enter the crank length (cm) [1]: 1
Enter the coupler length (cm) [3]: 3
Enter the rocker length (cm) [4]: 4
Enter the length or r5 (cm) [1.576]: 1.8
Enter the length or r6 (cm) [2.544]: 2.6
Enter the x coordinate of pivot G (cm) [1.075] 1.1
Enter the y coordinate of pivot G (cm) [3/615]: 3.6
Enter coupler point radius (cm) [1.579]: 1.6
Enter angle from coupler line to coupler point (deg) [30]: 45
Enter the frame angle (deg) [0]: 0
Enter the angular velocity of crank(rad/sec) [1]: 1
Enter the assembly mode for the dyad (+1 or -1) [1]: 1
I am working ...
repeat animation? y/n [y]: n
Repeat animation? y/n [y]: n
```

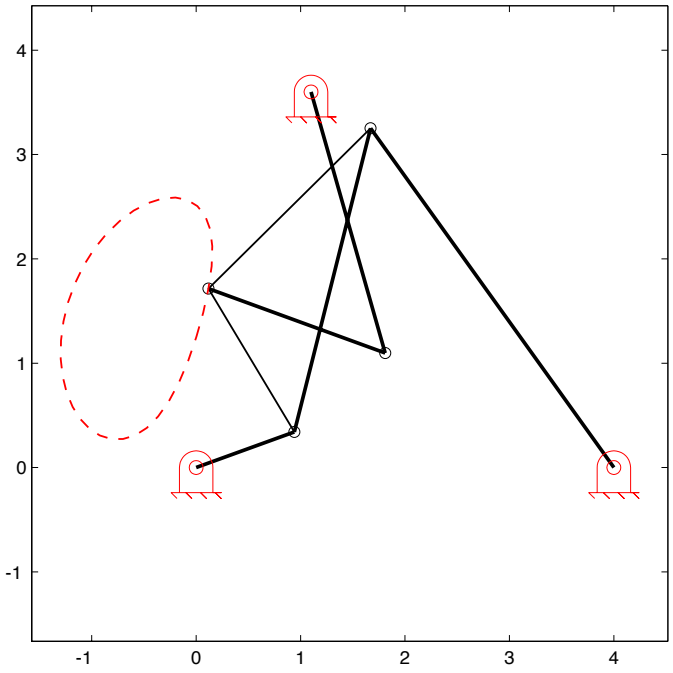


Fig. 4.F.3: Six-bar linkage in sample calculations.

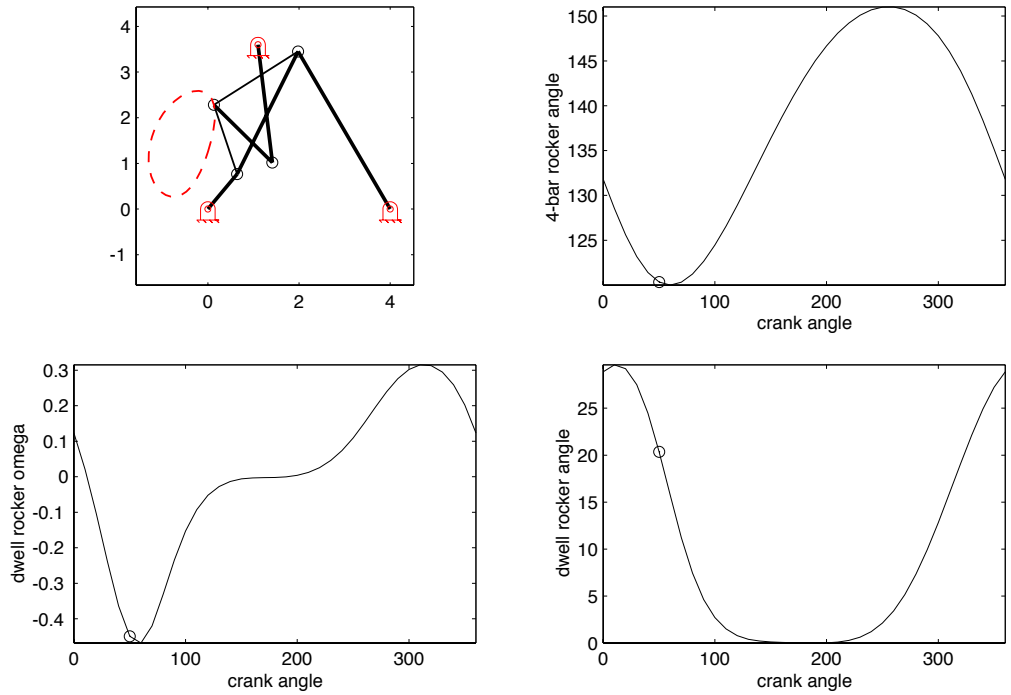


Fig. 4.F.4: Analysis of specific linkage selected.

4.G MATLAB Procedure for Generating Cognate Linkages

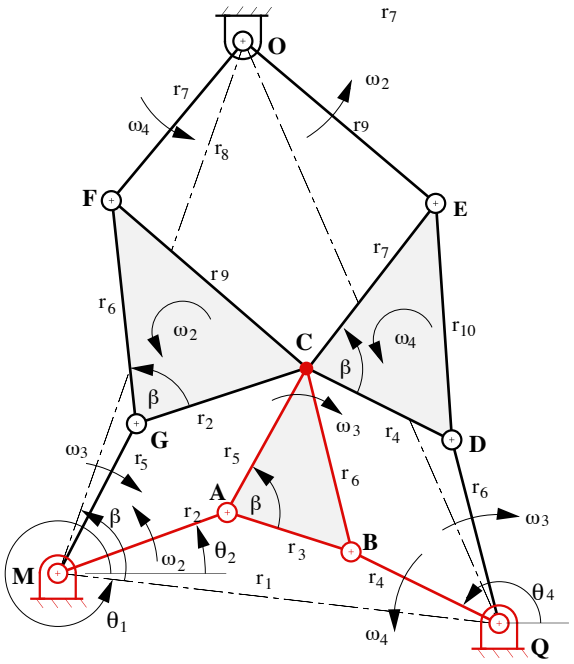


Fig. 4.G.1: Nomenclature for cognate linkages determined by program

4.G.1 Introduction

In this appendix, the equations for the cognate linkages given in Table 4.1 are programmed. The input to the program is the basic four-bar linkage geometry and the location of the coupler point C as shown in the MQ linkage of Fig. 4.G.1. The results from the analysis are the cognate linkages.

The MATLAB procedure involves the main routine (*cognates.m)* and the four routines *axisadjust.m, fourbar.m*, *bushing.m*, *circle.m* and *rbody1.m*. These routines have been described previously, and the names and purpose of the individual routines are given in the following:

axisadjust.m: Function routine to adjust the axis limits so that the viewport and window in Matlab 5.0 are similar to those in Matlab 4.2 when the command "axis equal" is used

circle.m: This function returns the coordinates for a circle given the center location and radius

bushing.m: This function returns the coordinates for drawing a bushing or frame pivot.

fourbar.m: This function analyzes a four-bar linkage for position, velocity, and acceleration.

rbody1.m: This function analyzes the third point on a rigid body when the kinematic information for two other points are given.

After the basic data are input based on the nomenclature given in Fig. 4.G.1, the mechanism is analyzed and animated for a full cycle of motion. Next the cognate linkages are determined and the three linkages are animated together. The final plots show separate drawings of the three linkages animated.

The results of a sample analysis are given in the following section.

4.G.2 Sample Run Using *cognates.m*

In the following, a copy of the input screen is given in Table 4.G.1 and the plots are displayed in Figs. 4.G.3 - 4.G.4.

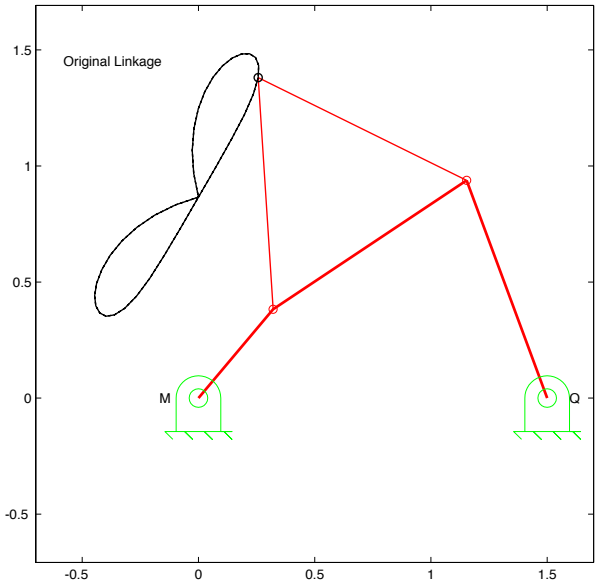


Fig. 4.G.2: Four-bar linkage in sample calculations.

Table 4.G.1: Input and output corresponding to sample analysis

```
         Cognate Linkage Analysis Program
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (cognate.dat): manual.dat
Enter number of cycles (rev) [3]: 3
Enter the frame length [2.75]: 1.5
Enter the crank length [0.9]: .5
Enter the coupler length [2.4]: 1
Enter the rocker length [1.3]: 1
Enter coupler point radius [4.85]: 1
Enter angle from coupler line to coupler point (deg) [-4]: 60
Enter the frame angle (deg) [57]: 0
Enter the assembly mode [+1 or -1] [-1]: -1
I am working ...

           Program results
     r1        r2        r3        r4        r5
   1.50      0.50      1.00      1.00      1.00

     r6        r7        r8        r9       r10
   0.50      1.00      1.50      0.50      1.00

Repeat animation? y/n [y]: n
```

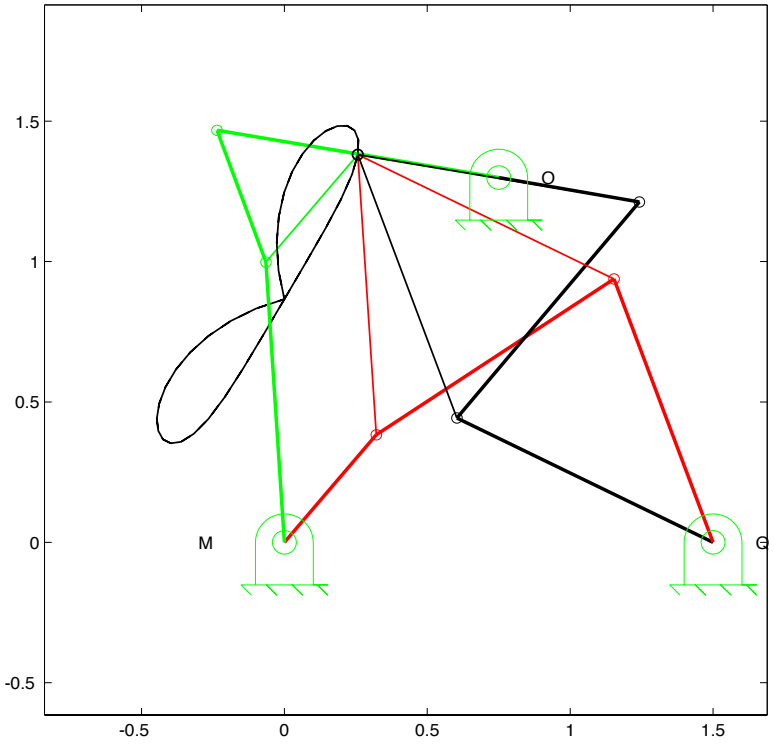


Fig. 4.G.3: Four-bar linkage cognates.

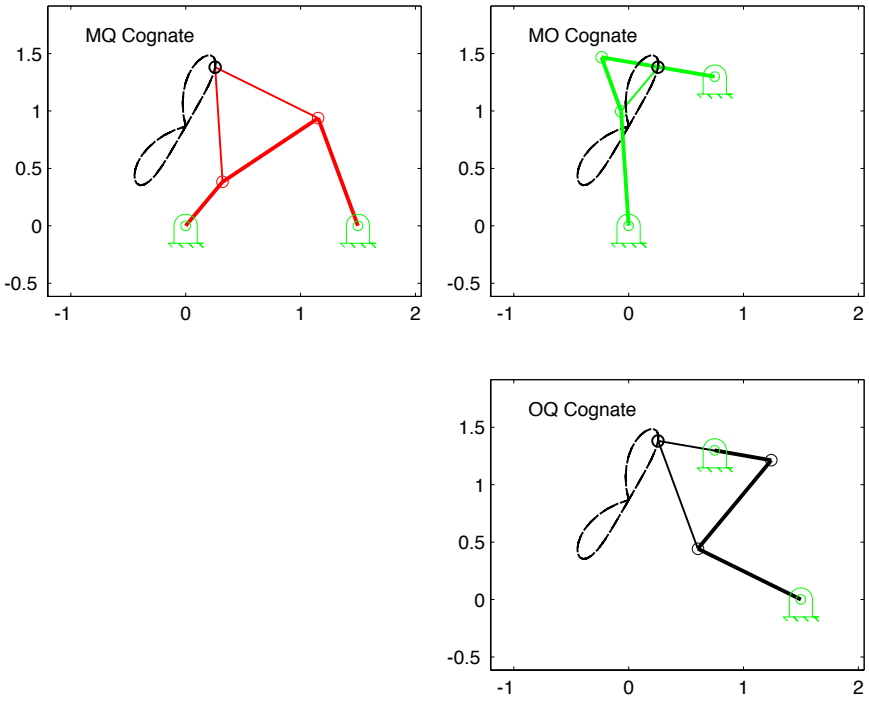


Fig. 4.G.4: Four-bar linkage cognates separated such that each generates the same coupler curve.

4.H MATLAB Procedure for Euler-Savary Equation

4.H.1 Introduction

The information in Appendix 4.H was originally contained in the main text book; however, it was removed because of limited space. The entire development is given here although the MATLAB programs apply to only part of what is presented.

Another way of generating a point path with desired properties is to use curvature theory. This provides a way of precisely controlling the trajectory in one position of a lamina. For example, the direction and curvature of the path of a given point can be controlled in a given position. The expectation is that the path will retain a similar curvature at all positions near to the designated point.

Curvature theory is actually closely related to the theory of motion generation through a series of finitely separated positions. It can be thought of as the limiting case in which the design positions become infinitesimally separated. There are many similarities. For example, as was shown in Section 4.2.5, the points in a lamina which lie on a straight line in three specified positions of that lamina lie on a circle. The corresponding result when the positions become infinitesimally separated is that at any instant in the motion of a lamina, the points whose paths have inflections, that is the points whose paths are locally straight, lie on a circle, called the inflection circle. The inflection circle passes through the instantaneous center and is tangent to the same line as the fixed and moving centrodes, which are the loci of the successive positions of the instantaneous center relative to the fixed and moving reference frames. The pole triangle collapses into the instantaneous center.

4.H.2 Two Infinitesimally Separated Positions

Specifying two design positions infinitesimally separated from one another is equivalent to specifying a position of a lamina and the velocity state of the lamina as it moves through that position. The velocity state can be specified by specifying the velocity $\boldsymbol{v}_O$ of the point O in the moving lamina which is instantaneously coincident with the origin, together with the angular velocity $\boldsymbol{\omega}$ of the lamina. The velocity of any other point A is then given by

$$\boldsymbol{v}_A = \boldsymbol{v}_O + \boldsymbol{v}_{A/O} = \boldsymbol{v}_O + \omega \times \boldsymbol{r}_{A/O} \tag{H.1}$$

where $\mathbf{r}_{A/O}$ is the directed line $\overline{OA}$.

Let us choose any point, C, in the moving lamina as a circle point. We seek a crank, with circle point at C such that the path of C produced by that crank is tangent to the path of C required by the velocity state. That is, the circular path of C produced by the crank should have the velocity vector $\boldsymbol{v}_C$ tangent to it. Clearly, any point on the normal to $\boldsymbol{v}_C$ through C can serve as the center point C*.

Example 4.H.1 (Synthesis of Linkage for Specified Velocity of Point in Coupler)

Problem:

Synthesize a four-bar linkage to give the coupler point at the origin a velocity of one unit per second in the *X* direction when the angular velocity is 4 rad/sec counter-clockwise.

Solution:

Let the four-bar linkage be defined in the usual manner with link 2 as the driver and link 3 as the coupler.

From the problem statement, point O_3 is the coupler point at the origin (coordinates relative to the frame are 0,0). In the following, the subscript 3 will be dropped because it is understood that all points being considered are in the coupler.

$$\boldsymbol{v}_O = 1\mathbf{i},\ \boldsymbol{\omega} = 4\boldsymbol{k} \text{ rad/sec}$$

For the four-bar linkage, we need to select two circle points, and for this we will choose points C(1, 1), and D(2, 0). Then,

$$\boldsymbol{v}_C = \boldsymbol{v}_O + \boldsymbol{v}_{C/O} = \boldsymbol{v}_O + \boldsymbol{\omega} \times \boldsymbol{r}_{C/O} = 1\boldsymbol{i} + 4\boldsymbol{k} \times (\boldsymbol{i} + \boldsymbol{j}) = -3\boldsymbol{i} + 4\boldsymbol{j}$$

where $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ are orthogonal unit vectors in the x, y and normal directions, respectively. Also,

$$\boldsymbol{v}_D = 1\ \boldsymbol{i} + 4\ \boldsymbol{k} \times (2\ \boldsymbol{i}) = \boldsymbol{i} + 8\,\boldsymbol{j}$$

Points C and D and velocities $\boldsymbol{v}_C$ and $\boldsymbol{v}_D$ are plotted on Fig. 4.H.1. The normals to $\boldsymbol{v}_C$ and $\boldsymbol{v}_D$ at those points were drawn and C* and D* were selected on those normals. The resulting linkage is C*CD D*. Compare this procedure to that used for two finitely separated positions.

Note that this linkage will give a different velocity state for each value of angular velocity for the coupler. Therefore, an infinite number of velocity states are possible.

The instant center for the coupler is shown in Fig. 4.H.2. Notice that C*CI and D D*I are collinear. This corresponds to the result that a crank subtends angle $\theta_{12}/2$ at the pole P_{12}. As θ_{12} approaches zero, the pole becomes co-linear with the circle and center points, and becomes the instantaneous center of rotation, I.

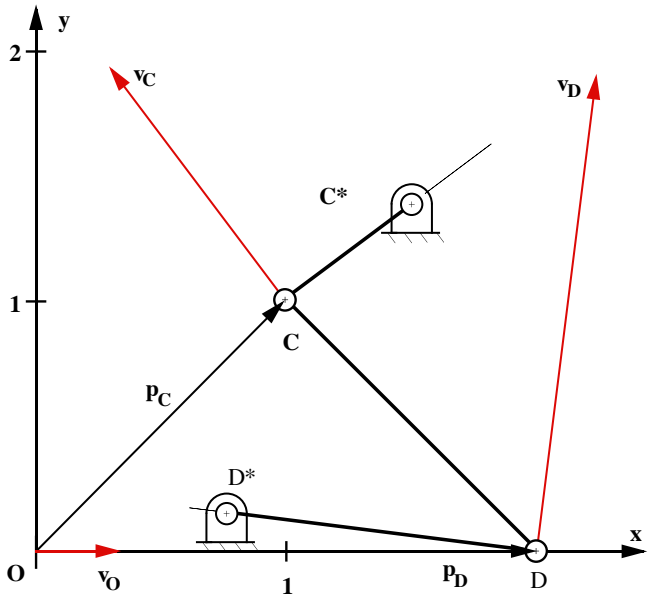


Fig. 4.H.1: The solution of Example 4.6.

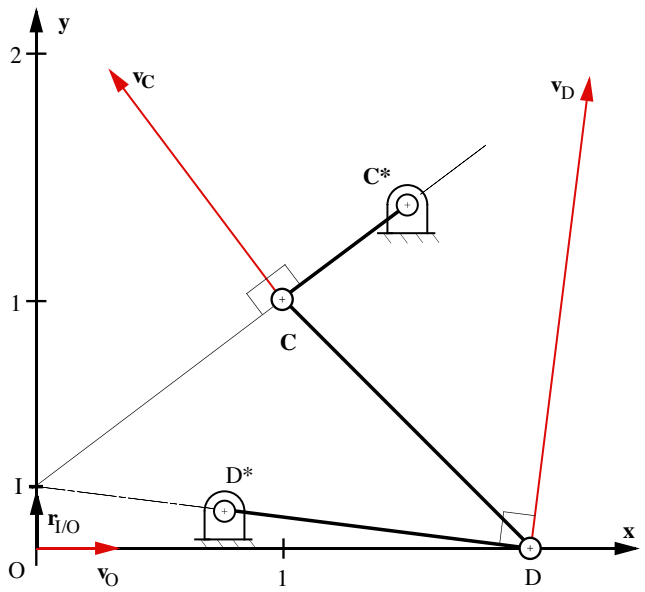


Fig. 4.H.2: The location of the instantaneous center for the velocity field of Example 4.6.

In the case of two finitely separated positions, we found that it was also possible to move the moving lamina through the two design positions using only a single pivot between the fixed and moving planes as shown in Fig. 4.14. This point was the pole for those two positions. Correspondingly, the required velocity state can be generated by means of a single pivot at the instantaneous center of the motion. The location, $\boldsymbol{r}_I$, of the instantaneous center relative to the origin is obtained from Eq. (H.1) by letting $\boldsymbol{v}_C = 0$ to get

$$0 = \boldsymbol{v}_O + \boldsymbol{\omega} \times \boldsymbol{r}_{I/O}$$

or

$$0 = \boldsymbol{\omega} \times \boldsymbol{v}_O + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r}_{I/O}) = \boldsymbol{\omega} \times \boldsymbol{v}_O - \omega^2 \, \boldsymbol{r}_{I/O}$$

or

$$\boldsymbol{r}_{I/O} = \frac{\boldsymbol{\omega} \times \boldsymbol{v}_O}{\omega^2} = \frac{\boldsymbol{k} \times \boldsymbol{v}_O}{\omega} \tag{H.2}$$

where $\boldsymbol{\omega} = \omega \boldsymbol{k}$, requiring that ω be positive counter-clockwise.

In Example 4.H.1 above,

$$\boldsymbol{r}_{I/O} = \frac{\boldsymbol{k} \times (1\boldsymbol{i})}{4} = \frac{\boldsymbol{j}}{4}.$$

This is shown in Fig. 4.H.2.

Notice that from Eq. (H.2), whatever solution is used

$$\frac{v_O}{\omega} = |\boldsymbol{r}_{I/O}|$$

is constant. That is, regardless of the angular velocity, the ratio of the velocity of the point at the origin to that angular velocity in the design position is constant. Put another way

$$\frac{d\boldsymbol{p}_O}{d\theta}$$

is constant, where $\boldsymbol{p}_O$ is the position vector from the origin of the coordinate system to the coupler point O which has coordinates (momentarily) of (0, 0). It is convenient to say that we are specifying the velocity state of the moving body, but it is more precise to say that we are specifying the derivative of the position of a point on the coupler with respect to the coupler angle.

4.H.3 Three Infinitesimally Separated Positions

4.H.3.1 Center of Curvature of Path of Moving Point Relative to Frame

Specifying three infinitesimally separated design positions is equivalent to specifying a position of the moving lamina and its velocity and acceleration states in that position. In addition to the velocity of the point in the moving lamina coincident with the origin and the angular velocity, we must specify the acceleration $\boldsymbol{a}_O$ of the point at the origin and the angular acceleration, $\boldsymbol{\alpha}$ of the moving lamina. The acceleration of any point, A, in the moving lamina can then be found

$$\boldsymbol{a}_A = \boldsymbol{a}_O + \boldsymbol{a}_{A/O} = \boldsymbol{a}_O + \alpha \times \boldsymbol{r}_{A/O} + \omega \times (\omega \times \boldsymbol{r}_{A/O}) = \boldsymbol{a}_O + \alpha \times \boldsymbol{r}_{A/O} - \omega^2 \boldsymbol{r}_{A/O} \tag{H.3}$$

Given the velocity and acceleration states of the moving lamina, we can find the radius of curvature of the path of any point in the moving lamina as that lamina passes through the design position. This is done by resolving the acceleration of that point into components tangent to, and normal to its path. Let $\boldsymbol{n}$ be a unit vector normal to the path that A traces on the frame and let $\mathbf{t}$ be a unit vector tangent to that path. We know that the velocity of the point will be tangent to the path that the point traces on the frame. Therefore, $\boldsymbol{t}$ is in the $\boldsymbol{v}_A$ direction, and $\boldsymbol{n}$ is directed so the $\boldsymbol{k} \times \boldsymbol{t} = \boldsymbol{n}$. Then the acceleration of point A can be written as:

$$\boldsymbol{a}_A = \boldsymbol{a}_A^t + \boldsymbol{a}_A^n = a_A^t \boldsymbol{t} + a_A^n \boldsymbol{n}$$

When the acceleration is expressed in terms of the normal and tangential components, it is the normal component which is a function of velocity and geometry. An expression for this component was derived in Section 2.5 when coincident points were considered. In particular, the acceleration of A can be rewritten as

$$\boldsymbol{a}_A = a_A^t \boldsymbol{t} + \frac{v_A^2}{\rho} \boldsymbol{n} \tag{H.4}$$

where ρ is the radius of curvature of the path that the point A traces on the frame. Equation (H.4) is derived in most undergraduate engineering mechanics texts, and a detailed derivation is given by Hall[1].

If we take the dot product of $\boldsymbol{n}$ with each side of Eq. (H.4), we get

$$\boldsymbol{n} \bullet \boldsymbol{a}_A = a_A^n = \frac{v_A^2}{\rho}$$

or

---

[1]Hall, A.S., *Kinematics and Linkage Design*. Balt Publishers, West Lafayette, IN (1961).

$$\rho = \frac{v_A^2}{\boldsymbol{n} \bullet \boldsymbol{a}_A} = \frac{v_A^2}{a_A^n} \qquad \text{(H.5)}$$

Equation (H.5) allows us to locate the center of curvature of the path of any point in a linkage once the basic velocity and acceleration analyses have been completed. The center of curvature of the path is in the direction of the normal component of acceleration. In Eq. (H.5), the normal component of acceleration can be plus or minus. If it is plus, it is in the ***+n*** direction, and if it is minus, it is in the ***-n*** direction.

Example 4.H.2 (Center of Curvature of the Path that a Point on the Coupler of a Slider-Crank Mechanism Traces on the Frame)

Problem:

Identify a procedure whereby we can locate the center of curvature of the path traced on the frame by points on the coupler of a slider-crank mechanism.

Solution:

Consider the slider-crank mechanism in Fig. 4.H.3, and assume that the path of C_3 is of interest. The center of curvature of the path is a purely geometric quantity, and therefore, the actual values used for the velocity and acceleration analysis are arbitrary. Also, the choice of the driver is arbitrary.

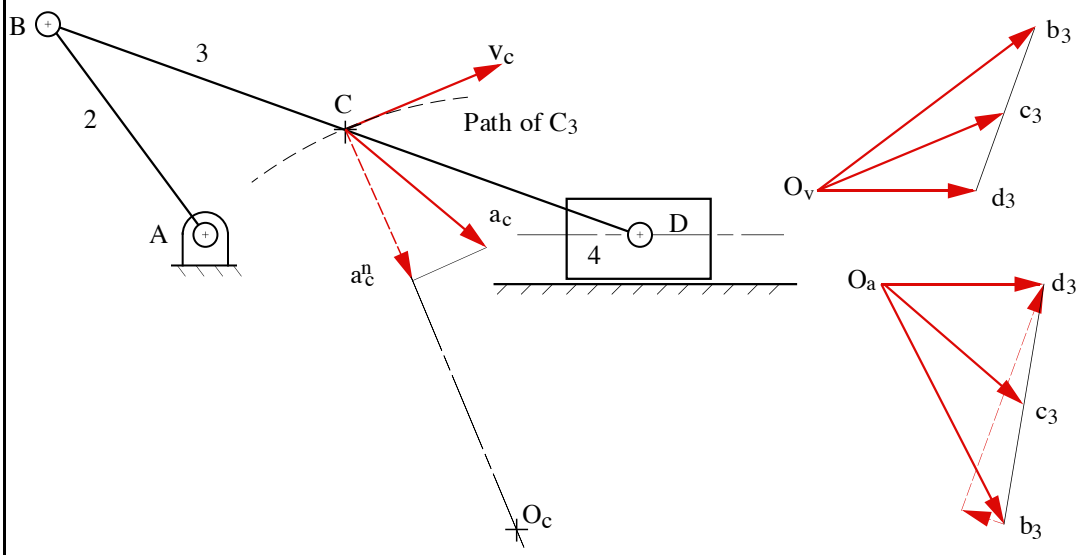


Fig. 4.H.3: Center of curvature of path of C_3 on the frame.

Because the velocity of C_3 is tangent to the path that C_3 traces on link 1, the velocity vector for C_3 indicates the direction of the tangent to the path. The center of curvature for the path will be on a line through C and normal to the velocity of C_3. From the acceleration analysis, we can determine ${}^1\boldsymbol{a}_{C_3}$ and resolve the acceleration into two components which are in the direction of ${}^1\boldsymbol{v}_{C_3}$ (tangent) and normal to ${}^1\boldsymbol{v}_{C_3}$. Then,

$$\boldsymbol{a}_{C3} = \boldsymbol{a}_{C_3}^n + \boldsymbol{a}_{C_3}^t$$

The radius of curvature of the path is calculated by using the magnitudes of the velocity and normal acceleration in the following:

$$r_{C/O_C} = \frac{v_{C3}^2}{a_{C3}^n}$$

The location of O_C is along the normal vector in the direction of $\boldsymbol{a}_{C3}^n$. This is shown in Fig. 4.H.3.

4.H.3.2 Synthesis Using the Center of Curvatureat a Point and Along a Path

To synthesize a linkage to move a lamina through three infinitesimally separated positions, we can take any point in that lamina, find the direction of its path and the radius of curvature of that path, and hence the center of curvature of the path. By locating the center point C* at the center of curvature, we get a crank which gives the required path direction and path curvature in the design position. Repeating this procedure for a second crank we generate a four-bar linkage which gives the required velocity and acceleration states while passing through the design position.

Example 4.H.3 (Synthesis of a four-bar linkage for three infinitesimally separated positions of a point in the coupler)

Problem:

The velocity state of a lamina is to be as in Example 4.h.2. That is, $\boldsymbol{v}_O$ = 1 in/s in the x direction, $\boldsymbol{\omega}$ = 4 rad/s counter-clockwise. In addition $\boldsymbol{a}_O$ is to be 20 in/s^2 in the y direction, and $\boldsymbol{\alpha}$ is to be 10 rad/s^2 clockwise.

Solution:

Choose C at position (1, 1) and D at (2, 0) as before, then $\boldsymbol{v}_C = -3\boldsymbol{i} + 4\boldsymbol{j}$ in/s and $\boldsymbol{v}_D = \boldsymbol{i} + 8\boldsymbol{j}$ in/s. From the problem statement, $\boldsymbol{\alpha} = -10\boldsymbol{k}$ rad/s^2 and $\boldsymbol{a}_o = 10\boldsymbol{j}$. Therefore, applying Eq. (H.3) gives:

$$\boldsymbol{a}_C = 10\boldsymbol{j} + (-10)\boldsymbol{k} \times (\boldsymbol{i} + \boldsymbol{j}) - 16(\boldsymbol{i} + \boldsymbol{j}) = -6\boldsymbol{i} - 16\boldsymbol{j}$$

At point C in the coupler,

$$\boldsymbol{t} = \frac{-3\boldsymbol{i} + 4\boldsymbol{j}}{\sqrt{3^2 + 4^2}} = -\frac{3}{5}\boldsymbol{i} + \frac{4}{5}\boldsymbol{j}$$

and

$$\boldsymbol{n} = \boldsymbol{k} \times \boldsymbol{t} = -\frac{3}{5}\boldsymbol{j} - \frac{4}{5}\boldsymbol{i}$$

so, applying Eq. (H.5)

$$\rho_C = \frac{3^2 + 4^2}{\left(-\frac{4}{5}\boldsymbol{i} - \frac{3}{5}\boldsymbol{j}\right) \bullet (-6\boldsymbol{i} - 16\boldsymbol{j})} = 1.736$$

Applying Eq. (H.3):

$$\boldsymbol{a}_D = 10\boldsymbol{j} + (-10)\boldsymbol{k} \times 2\boldsymbol{i} - 16\,(2\boldsymbol{i}) = -32\boldsymbol{i} - 10\boldsymbol{j}$$

At D

$$\boldsymbol{t} = \frac{\boldsymbol{i} + 8\boldsymbol{j}}{\sqrt{1^2 + 8^2}}$$

and

$$\boldsymbol{n} = \boldsymbol{k} \times \frac{1}{\sqrt{65}}(\boldsymbol{i} + 8\boldsymbol{j}) = \frac{1}{\sqrt{65}}(\boldsymbol{j} - 8\boldsymbol{i})$$

Applying Eq. (H.5):

$$\rho_D = \frac{1^2 + 8^2}{\frac{1}{\sqrt{65}}(\boldsymbol{j} - 8\boldsymbol{i}) \bullet (-32\boldsymbol{i} - 10\boldsymbol{j})} = 2.130$$

Using this data, C* and D* are located as shown in Fig. 4.H.4. Note that a minus sign on either ρ_C or ρ_D would indicate that the center of curvature is located in the -$\boldsymbol{n}$ direction.

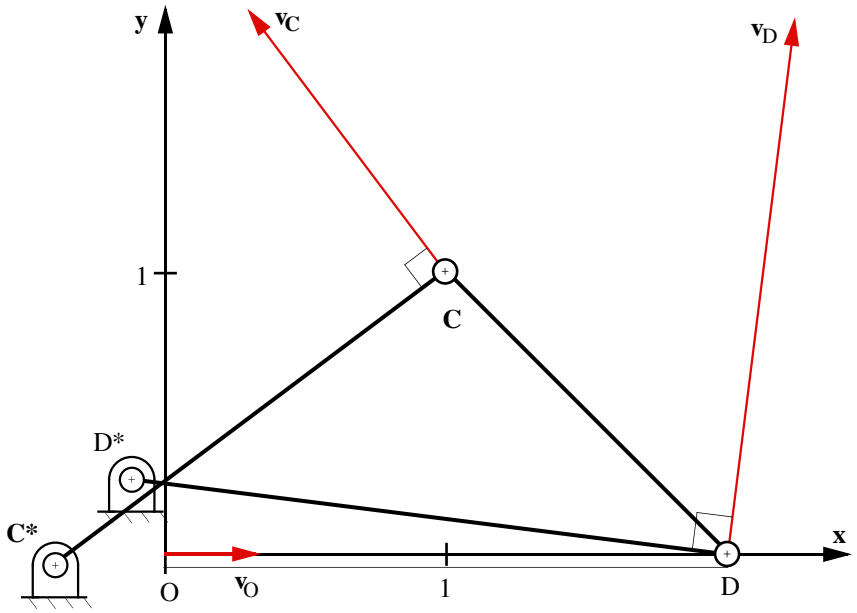


Figure 4.H.4: The solution to Example 4.H.3. C* and D* are selected to give the desired velocity and acceleration fields.

4.H.3.3 Inflection Circle

We found that, for three finitely separated positions, there are an infinite number of points whose three positions all lie on the same straight line and that they are distributed on a circle which passes through all three image poles. Let us seek the equivalent result for 3 infinitesimally separated positions: namely, the locus of points which, for a given velocity and acceleration state, have paths with locally infinite radius of curvature. Another way of stating this is the locus of points whose paths have points of inflexion at the instant of passing through the design position.

Looking at Eq. (H.5), we see that ρ approaching infinity implies $\boldsymbol{n} \bullet \boldsymbol{a}_A = 0$. In the general case, $\boldsymbol{a}_A$ will be nonzero. Then, since $\boldsymbol{n}$ is normal to $\boldsymbol{v}_A$, this implies that $\boldsymbol{v}_A$ and $\boldsymbol{a}_A$ have the same or opposite directions. Hence

$$\boldsymbol{v}_A \times \boldsymbol{a}_A = 0$$

Applying Eqs. (H.1) and (H.3)

$$(\boldsymbol{v}_O + \omega\,\boldsymbol{k} \times \boldsymbol{r}_{A/O}) \times (\boldsymbol{a}_O + \alpha\,\boldsymbol{k} \times \boldsymbol{r}_{A/O} - \omega^2 \boldsymbol{r}_{A/O}) = 0 \tag{H.6}$$

For the analysis, we may select the origin of coordinates to be at any location that we like. It will simplify the results if we move the origin to the instantaneous center, I, between the moving body and the frame. Then $\boldsymbol{v}_O$ becomes zero, and $\boldsymbol{a}_O = \boldsymbol{a}_I$. That is, $\boldsymbol{a}_O$ becomes the acceleration of the point in the moving body which is at the instantaneous center. Equation (H.6) then becomes

$$(\omega\boldsymbol{k} \times \boldsymbol{r}_{A/I}) \times (\boldsymbol{a}_I + \alpha\,\boldsymbol{k} \times \boldsymbol{r}_{A/I} - \omega^2 \boldsymbol{r}_{A/I}) = 0$$

or

$$(\boldsymbol{k} \times \boldsymbol{r}_{A/I}) \times \boldsymbol{a}_I - \omega^2\,(\boldsymbol{k} \times \boldsymbol{r}_{A/I}) \times \boldsymbol{r}_{A/I} = 0$$

or

$$(\boldsymbol{k} \times \boldsymbol{r}_{A/I}) \times \boldsymbol{a}_I + \omega^2 (r_{A/I})^2 \boldsymbol{k} = 0$$

Let the angle between $\boldsymbol{a}_I$ and $\boldsymbol{r}_{A/I}$ be γ_A (see Fig. 4.H.5), where γ_A is measured from $\boldsymbol{a}_I$ to $\boldsymbol{r}_{A/I}$. Then

$$(\boldsymbol{k} \times \boldsymbol{r}_{A/I}) \times \boldsymbol{a}_I = - r_{A/I} a_I \sin(\gamma_A + \pi/2)\,\boldsymbol{k} \tag{H.7}$$

since $|\boldsymbol{k} \times \boldsymbol{r}_{A/I}| = r_{A/I}$ and the angle between $\boldsymbol{k} \times \boldsymbol{r}_{A/I}$ and $\boldsymbol{a}_I$ is $-(\gamma_A + \pi/2)$. Hence

$$\omega^2 (r_{A/I})^2 \boldsymbol{k} - a_I\, r_{A/I} \cos\gamma_A\, \boldsymbol{k} = 0$$

or

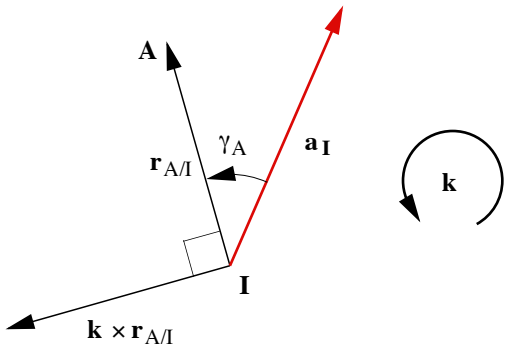


Fig. 4.H.5: The geometry of the vectors in Eq. (H.7).

$$r_{A/I} = \frac{a_I}{\omega^2}\cos\gamma_A \tag{H.8}$$

This is the equation of a circle passing through I with diameter

$$D = \frac{a_I}{\omega^2}. \tag{H.9}$$

The center of the circle through I is located on a line from I and in the $\boldsymbol{a}_I$ direction. This circle is called the *inflection circle* and is represented in Fig. 4.H.6. The inflection circle can be viewed as the limit of the image pole circle for three finitely separated positions as those positions become infinitesimally close. Just as the image pole circle (circle of sliders) is the locus of circle points whose three positions lie on a straight line, the inflection circle is the locus of points whose paths are locally straight.

We now seek an expression for the radius of curvature of the path of any point, A, in terms of the variables

used in Eq. (H.7). Equation (H.5) gives

$$\rho = \frac{v_A^2}{\boldsymbol{n} \bullet \boldsymbol{a}_A}$$

Substituting from Eqs. (H.1) and (H.3) for $\boldsymbol{v}_A$ and $\boldsymbol{a}_A$ with origin at the instantaneous center

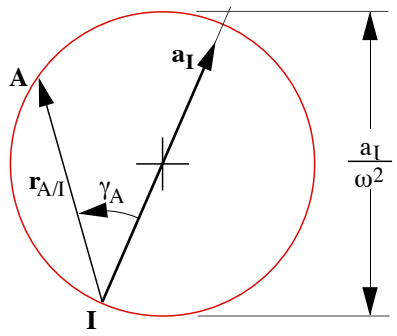


Fig. 4.H.6: The inflection circle, for given I, a_I, r, ω, and γ_A.

$$\boldsymbol{v}_A = 0 + \omega \boldsymbol{k} \times \boldsymbol{r}_{A/I}, \quad v_A{}^2 = \omega^2 \, r_{A/I}^2$$

Now

$$\mathbf{n} = \mathbf{k} \times \frac{\mathbf{v}_A}{v_A} = -\frac{\omega \mathbf{r}_{A/I}}{\omega r_{A/I}} = -\frac{\mathbf{r}_{A/I}}{r_{A/I}}$$

and from Eq. (H.3),

$$\mathbf{a}_A = \mathbf{a}_I + \alpha \, \mathbf{k} \times \mathbf{r}_{A/I} - \omega^2 \mathbf{r}_{A/I}$$

so

$$\boldsymbol{n} \bullet \boldsymbol{a}_A = -\frac{\boldsymbol{r}_{A/I}}{r_{A/I}} \bullet \boldsymbol{a}_I + 0 + \omega^2 r_{A/I}$$

Referring to Fig. 4.H.6

$$\boldsymbol{r}_{A/I} \bullet \boldsymbol{a}_I = r_{A/I} a_I \cos\gamma_A$$

so

$$\boldsymbol{n} \bullet \boldsymbol{a}_A = -a_I \cos\gamma_A + \omega^2 r_{A/I}$$

and

$$\rho = \frac{\omega^2 r_{A/I}^2}{\omega^2 r_{A/I} - a_I \cos\gamma_A}$$

Now, if D is the diameter of the inflection circle, Eq. (H.9) gives

$$D = \frac{a_I}{\omega^2}$$

and so

$$\rho = \frac{r_{A/I}^2}{r_{A/I} - D\cos\gamma_A} \tag{H.10}$$

Equation (H.10) is one form of the Euler-Savary equation. The Euler-Savary Equation is very useful because, given the instantaneous center and inflection circle, it can be used to locate the center point corresponding to any given circle point, or vice-versa. The inflection circle is readily constructed for a given four-bar linkage, and it is, therefore, more convenient to work with the inflection circle than with the variables ω and $\boldsymbol{a}_I$.

The geometric meaning of the Euler-Savary Equation is discernible by referring to Fig. 4.H.7. Let A be the point whose path curvature is sought. If we use directed line segments, $r_{A/I}$ points from I to A, and $r_{A/A*}$ points from A^* to A. Also, $r_{JA/I} = D\ cos\gamma_A$ where J_A is the location where a ray from I to A crosses the inflection circle. Hence, if A^* is the center of curvature of the path of point A, then $\rho = r_{A/A*}$ and

$$r_{A/A^*} = \frac{r_{A/I}^2}{r_{A/I} - r_{JA/I}}.$$

Now $r_{A/I} - r_{Q/I} = r_{A/Q}$ so

$$\frac{r_{A/A^*}}{r_{A/I}} = \frac{r_{A/I}}{r_{A/JA}} \tag{H.11}$$

which can be viewed as the geometric form of the Euler-Savary Eq. (H.10).

4.H.3.4 Different Forms for the Euler-Savary Equation

The Euler-Savary Equation can be expressed in several different ways, and the different forms are useful depending on the known quantities when a problem is formulated. For example, another form can be derived from Eq. (H.11) as follows:

$$r_{A/A^*} = r_{A/I} + r_{I/A^*} = \frac{r_{A/I}^2}{r_{A/JA}} = \frac{r_{A/I}^2}{r_{A/I} + r_{I/JA}}$$

or

$$(r_{A/I} + r_{I/A*})(r_{A/I} + r_{I/JA}) = r_{A/I}^2 + (r_{A/I})(r_{I/JA}) + (r_{I/A*})(r_{A/I}) + (r_{I/A*})(r_{I/JA}) = r_{A/I}^2$$

Simplifying

$$(r_{A/I})(r_{I/JA}) + (r_{I/A*})(r_{A/I}) + (r_{I/A*})(r_{I/JA}) = 0$$

Now division by $(r_{A/I})(r_{I/JA})(r_{I/A*})$ gives

$$\frac{1}{r_{I/A*}} + \frac{1}{r_{I/JA}} + \frac{1}{r_{A/I}} = 0$$

or

$$\frac{1}{r_{JA/I}} = \frac{1}{r_{A/I}} - \frac{1}{r_{A*/I}} \tag{H.12}$$

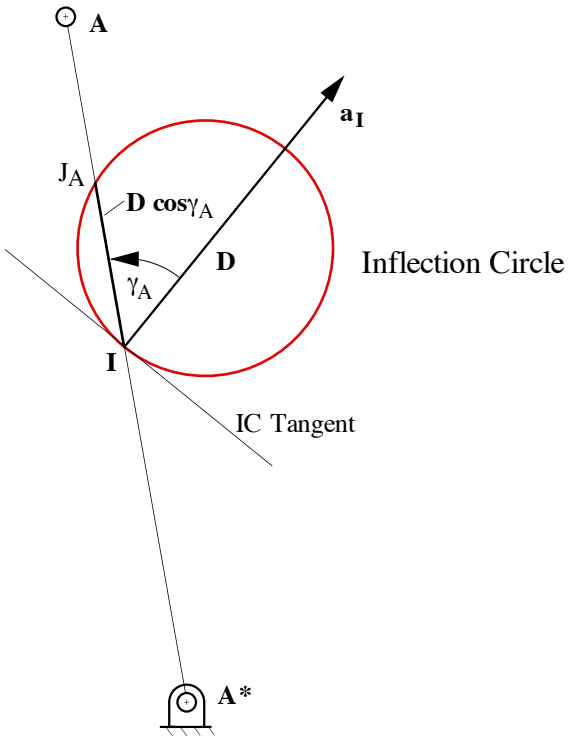


Fig. 4.H.7: The geometric relationship of the inflection circle with a center and circle point pair A*A.

Some of the different forms for the Euler-Savary Equation are summarized in Table 4.H.1. The terms used in Table 4.H.1 are shown schematically in Fig. 4.H.8. Most of the forms can be derived directly from Eq. (H.11) as was done in the case of Eq. (H.12); however, several of the forms are based on O_M and O_F, the centers of curvature of the moving and fixed centrodes corresponding to the instant center. These forms are derived by Hall[2]. Each form of the equation is based on a

[2]Hall, A.S., *Kinematics and Linkage Design*. Balt Publishers, West Lafayette, IN (1961).

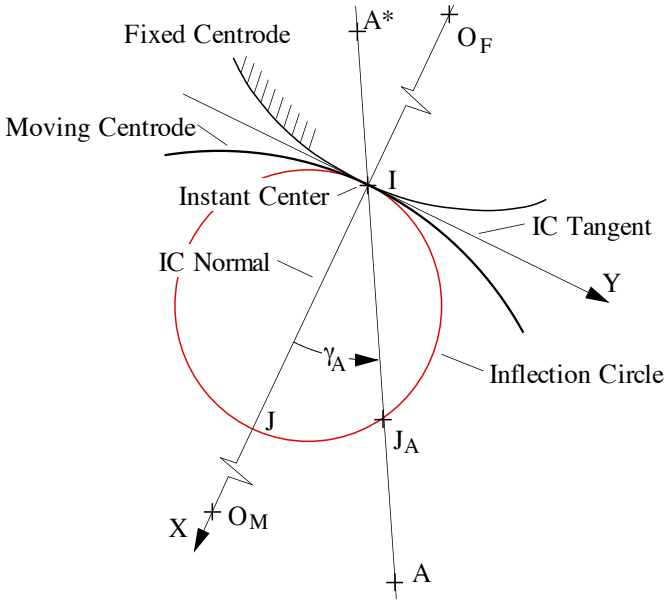


Fig. 4.H.8: Summary of terms for Euler-Savary equation

Table 4.H.1 Summary of forms of Euler-Savary equation

Using the ray I-A, different forms of the Euler-Savary Equation are:

$$r_{A/A^*} = \frac{r_{A/I}^2}{r_{A/JA}} \qquad \frac{1}{r_{OM/I}} - \frac{1}{r_{OF/I}} = \frac{1}{r_{J/I}}$$

$$r_{A/A^*} = \frac{r_{A/I}^2}{r_{A/I} - r_{JA/I}} \qquad \frac{1}{r_{JA/I}} = \frac{1}{r_{A/I}} - \frac{1}{r_{A^*/I}}$$

$$r_{A/A^*} = \frac{r_{A/I}^2}{r_{A/I} - r_{J/I}\cos\gamma_A} \qquad r_{A/I} = \frac{r_{JA/I}\, r_{A^*/I}}{r_{JA/I} + r_{A^*/I}}$$

$$\frac{1}{r_{OM/I}} - \frac{1}{r_{OF/I}} = \left(\frac{1}{r_{A/I}} - \frac{1}{r_{A^*/I}}\right)\cos\gamma_A \qquad r_{JA/I} = r_{A/I} - \frac{r_{A/I}^2}{r_{A/A^*}^2} r_{A/A^*}$$

single ray through the instant center *I*. Therefore, relative to the ray, each vector can be treated as a signed (±) number. One direction from I can be taken arbitrarily as positive; distances in the other direction are automatically taken as negative. Examples of different locations of circle points and center points are shown in Fig. 4.H.9.

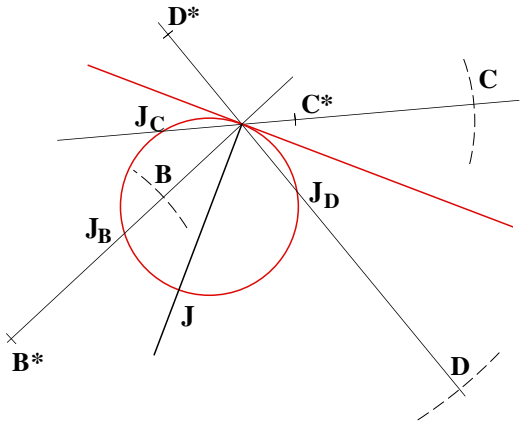


Fig. 4.H.9: Locations for different points according to the Euler-Savary equation.

Example 4.H.4 (Locating the Inflection Circle for Four-Bar Linkage)

Problem:

Locate the inflection circle for the four-bar linkage shown in Fig. 4.H.10.

Solution:

To find the inflection circle, we need to find three points lying on it. Three points which can be found from the information given are *I*, J_A, and J_B. First locate the instant center *I*. From Chapter 3, the location is where an extension of AA^* intersects the line defined by BB^*. Next find J_A. This can be found by rewriting Eq. (H.11) as

$$r_{A/JA} = \frac{r_{A/I}^2}{r_{A/A^*}} \tag{H.13}$$

From the geometry given in Fig. 4.H.10, $r_{A/I} = AB\sin(30^\circ) = 2$. Substituting numbers into Eq. (H.13), $r_{A/JA} = \frac{r_{A/I}^2}{r_{A/A^*}} = \frac{2^2}{4} = 1$ in the direction of r_{A/A^*}. This locates J_A between *A* and *A**. Next compute J_B using

$$r_{B/J_B} = \frac{r_{B/I}^2}{r_{B/B^*}}$$

From the geometry given in Fig. 4.H.10, $r_{B/I} = AB\cos(30^\circ) = 2\sqrt{3}$. Substituting numbers into Eq. (H.13) again gives $r_{B/J_B} = \frac{r_{B/I}^2}{r_{B/B^*}} = \frac{(2\sqrt{3})^2}{4\sqrt{3}} = \sqrt{3}$ in the direction of r_{B/B^*}. This locates J_B between *B* and *B** also. Given *I*, J_A, and J_B, the inflection circle can be drawn as shown in Fig. 4.H.10.

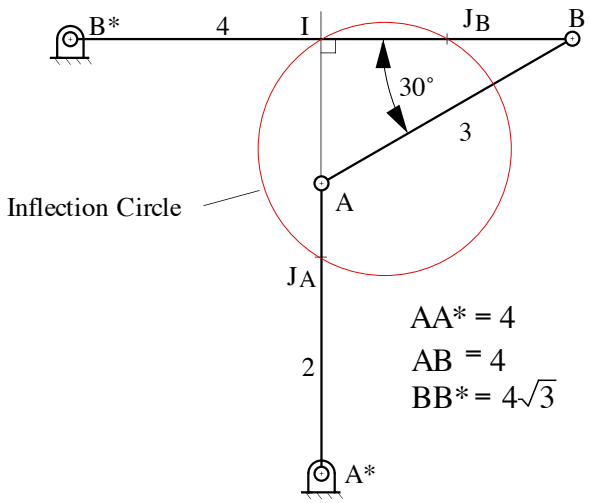


Fig. 4.H.10: Inflection circle for four-bar linkage in Example 4.H.4

Example 4.H.5 (Inflection Circle for Slider-Crank Mechanism)

Problem:

Locate the inflection circle for the slider-crank mechanism shown in Fig. 4.H.11. The link dimensions are $AA^* = 2$ m and $AB = 4$ m.

Solution:

Again, to find the inflection circle, we need to find three points lying on it. Three points which can be found from the information given are I, J_A, and J_B. First locate the instant center I using the procedure given in Chapter 3. The distance AB is given by

$$AB = 2\cos 30 + \sqrt{1^2 + 2^2} = 3.968$$

and $r_{A^*/I}$ is given by

$$r_{A^*/I} = 3.968 / \cos 30 = 4.582.$$

Also,

$$r_{A/I} = 4.582 - 2 = 2.582$$

and

$$r_{B/I} = (r_{A/I} + r_{A/A^*}) * \sin(30^\circ) = 2.291$$

Next find J_A using Eq. (H.13). For the values given, $r_{A/JA} = \dfrac{r_{A/I}^2}{r_{A/A^*}} = \dfrac{2.582^2}{2} = 3.333$ in the direction of r_{A/A^*}. Therefore, A^* is between A and J_A as shown in Fig. 4.H.11.

Next compute J_B. From Eq. (H.13), $r_{B/JB} = \dfrac{r_{B/I}^2}{r_{B/B^*}} = \dfrac{2.291^2}{\infty} = 0$. That is, J_B is located at B. We could have determined this by inspection because point B traces a straight path on the frame. Therefore, B must be on the inflection circle by definition. Given I, J_A, and J_B, the inflection circle can be drawn as shown in Fig.

4.H.11.

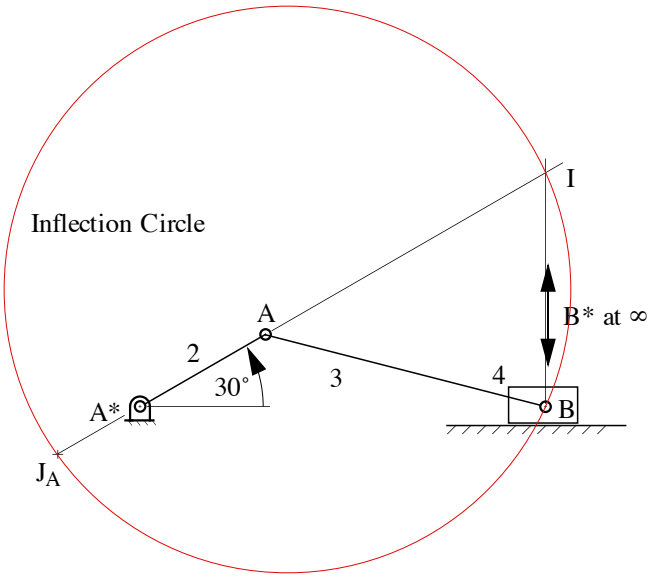


Fig. 4.H.11: Inflection circle for slider-crank mechanism in Example 4.H.4.

Example 4.H.5 (Inflection Circle and Radius of Curvature)

Problem:

Determine the radius of curvature of the path that point C_3 in Fig. 4.H.12 traces on the frame. Link 3 rolls on link 4 without slipping. The dimensions for the linkage are as follows: $AA^* = 1$ cm, $B^*A^* = 1$, $AC = 2$ cm, and the radius of the roller is 0.2 cm.

Solution:

To solve the problem, we first need to find the inflection circle. As in Examples 4.H.2 and 4.H.3, we need to find three points lying on the inflection circle to define it. Three points which can be found from the information given are I, J_A, and J_B. Point B_3 is not indicated directly on the drawing, however, we can locate B_3 by visualizing the path that B* traces on link 3. That path is a straight line; therefore, the center of curvature of the path is at infinity. Points B and B^* switch roles when we invert the motion and make link 3 the reference and allow the frame to move. Thus, B_3 is the center of curvature of the path of B^* relative to link 3, and B^* is the center of curvature of the path of B_3 relative to the frame. Therefore, in this problem, B_3 is at infinity in the direction indicated in Fig. 4.H.12.

Locate the instant center, I, by finding the intersection of the rays through BB^* and AA^*. To find the intersection, the angle ϕ is required. From geometry, this is given by

$$\phi = \cos^{-1}\left(\frac{0.2}{1.414}\right) = 81.869^\circ$$

Then,

$$r_{A^*/I} = r_{A^*/B^*}\tan(\phi - 45) = 1\tan(36.869) = 0.750$$

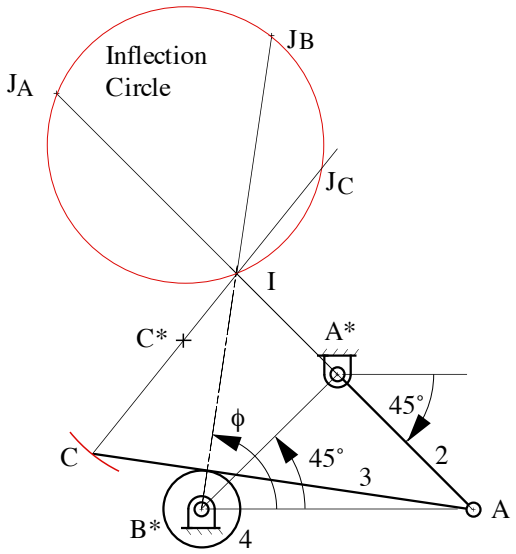


Fig. 4.H.12: Inflection circle for Example 4.11

Next find J_A. using Eq. (H.13). That is, $r_{A/JA} = \dfrac{r_{A/I}^2}{r_{A/A^*}} = \dfrac{(1+0.75)^2}{1} = 3.062$ in the direction of r_{A/A^*}. Therefore, A^* is between A and J_A as shown in Fig. 4.H.12.

To find the location of J_B, we cannot use Eq. (H.13) because B is at infinity. Instead, we can use the form of the equation given by Eq. (H.12). That is,

$$\frac{1}{r_{JB/I}} = \frac{1}{r_{B/I}} - \frac{1}{r_{B^*/I}}$$

or

$$\frac{1}{r_{JB/I}} = \frac{1}{\infty} - \frac{1}{r_{B^*/A^*}/\cos(\phi - 45)}$$

or

$$r_{JB/I} = -r_{B^*/A^*}/\cos(\phi - 45) = -1/\cos(36.869) = -1.250$$

or 1.250 in the opposite direction of $r_{B^*/I}$. Therefore, I is between B^* and J_B as shown in Fig. 4.H.12.

To locate the center C^*, we must first find J_C by drawing a ray from C through I as shown in Fig. 4.H.12. We can measure $r_{I/JC}$ directly to be 0.711. Also, $r_{C/I} = 1.197$. Then from Eq. (H.13),

$$r_{C/JC} = \frac{r_{C/I}^2}{r_{C/C^*}}$$

or $r_{C/C^*} = \dfrac{r_{C/I}^2}{r_{C/JC}} = \dfrac{(1.197)^2}{(1.197+0.711)} = 0.752$ in the same direction as $r_{I/JC}$

Therefore, C* is between C and I. The location is shown in Fig. 4.H.12. The approximate path of C is also drawn in Fig. 4.H.12.

4.H.4 Relationship Among IC, Centrodes, IC Tangent, and IC Velocity

The relative motion between two rigid bodies is equivalent to two curves called centrodes rolling on each other as discussed in Chapter 2. One centrode is fixed to one body, and the second is fixed to the other body. This is represented in Fig. 4.H.13 for the coupler of a four-bar linkage.

The point of contact is the instant center, and the centrodes are the paths of the instant centers on the two bodies. The instant center (IC) tangent is the common tangent to the two centrodes. The IC velocity is the instantaneous velocity with which the IC shifts; it is along the IC tangent. Note that the point which has the IC velocity will belong to neither of the rigid bodies being considered. Relative to the two bodies, the IC is at a different location for each relative position of the two bodies. This situation is shown in Fig. 4.H.14. In that figure, the instant center I_{13} is in a different location relative to links 1 and 3 for each position of the linkage. The path of the instant center is defined by the path of point I_5 relative to the frame where link 5 is the ball captured between the two yokes in Fig. 4.H.14. This path will be the fixed centrode. For any instantaneous position, the location of point I_5 coincides with the instant center, I_{13}, and the velocity of I_5 is the IC velocity discussed above.

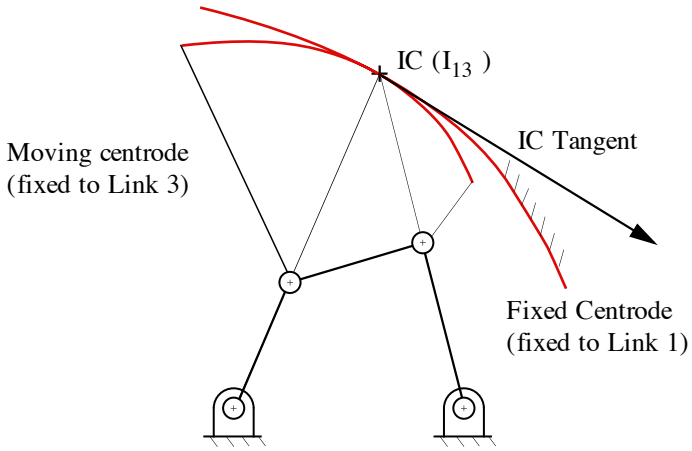


Fig. 4.H.13: Location of instant center I_{13} and centrodes for a four-bar linkage.

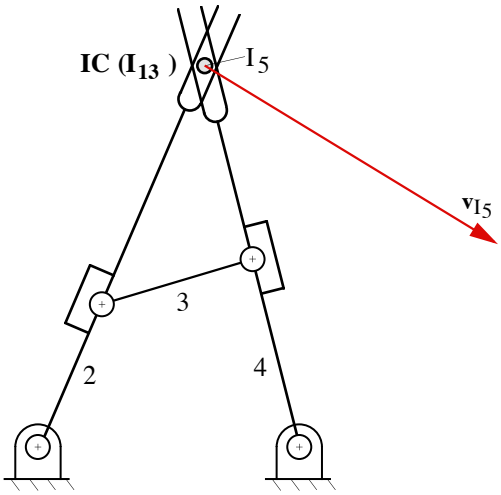


Fig. 4.H.14: The path of I_{13} can be traced by I_5 as shown. Here, link 5 is the ball captured by the two yokes on links 2 and 4.

4.H.5 Analytical Form for Euler Savary Equation

The approach used in Examples 4.H.3 - 4.H.5 uses one of the forms of the Euler-Savary Equation given in Table 4.H.1. These equations lend themselves to the graphical solution of the Euler-Savary Equation. To use the equations, we must establish a positive direction and identify that direction in the calculations. When programming the equations, it is convenient to work initially with points or absolute vectors rather than relative vectors. From the absolute vectors, the vectors in Table 4.H.1 can be established. For example,

$$r_{A/A^*} = |\boldsymbol{r}_{A/A^*}| = |\boldsymbol{r}_A - \boldsymbol{r}_{A^*}| \qquad r_{A/I} = |\boldsymbol{r}_{A/I}| = |\boldsymbol{r}_A - \boldsymbol{r}_I|$$

$$r_{A/JA} = |\boldsymbol{r}_{A/JA}| = |\boldsymbol{r}_A - \boldsymbol{r}_{JA}| \qquad r_{J/I} = |\boldsymbol{r}_{J/I}| = |\boldsymbol{r}_J - \boldsymbol{r}_I|$$

etc. With these substitutions, the equations in Table 4.H.1 can be programmed easily to compute the unknowns. MATLAB routines for the most common calculations are on the disk included with this book. Combinations of these routines can be used to write programs for finding the inflection circle and determining the center of curvature of selected points on different links. A routine for making these calculations are given for a four-bar linkage.

4.H.6 The Bobillier Constructions

As indicated in Example 4.H.3, if we have a four-bar linkage, we can determine the inflection circle by locating J_A and J_B. However, calculations are required to locate these two points. The Bobillier constructions allow the inflection circle to be determined without calculations. The Bobillier constructions are graphical solutions of the Euler-Savary equation for a four-bar linkage. That is, they permit the location of the center point corresponding to a given circle point for three infinitesimally separated positions.

4.H.6.1 Bobillier's Theorem

Bobillier's theorem states that the angle between the centrode tangent at the instantaneous center of the coupler relative to the base of a four-bar linkage and one of the cranks is equal to the angle between the other crank and the collineation axis. The collineation axis is the line joining the instantaneous center of the coupler relative to the base to the instantaneous center of one crank relative to the other as shown in Fig. 4.H.15. This theorem permits easy location of the centrode tangent. A line normal to the centrode tangent at the instant center gives a locus for the center of the inflection circle.

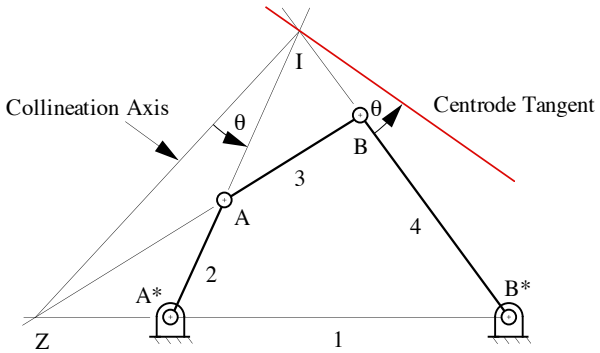


Fig. 4.H.15: Statement of Bobillier's Theorem. The theorem states that the angles marked θ are equal.

Proof

For a four-bar linkage, we can find the IC tangent for I_{13} by a simple relative velocity analysis. Referring

to Fig. 4.H.16, let the instant center location for I_{13} be designated simply as I. Also, let I_5 be a point on rigid body 5 which traces the path of the instant center as shown in Figs. 4.H.14 and 4.H.16. Then the following relationships apply:

$$^1\boldsymbol{v}_{I5} = {}^1\boldsymbol{v}_{I2} + {}^1\boldsymbol{v}_{I5/I2} = {}^1\boldsymbol{v}_{I4} + {}^1\boldsymbol{v}_{I5/I4}$$

$$^1\boldsymbol{v}_{I4/I2} = {}^2\boldsymbol{v}_{I4/I2} = {}^2\boldsymbol{v}_{I4/\text{anypt. in system2}}$$

$$= {}^2\boldsymbol{v}_{I4/\text{anypt. in system 4 withzero velocity relative to System 2}} = {}^2\boldsymbol{v}_{I4/(I24)2} = {}^2\boldsymbol{v}_{I4/(I24)4}$$

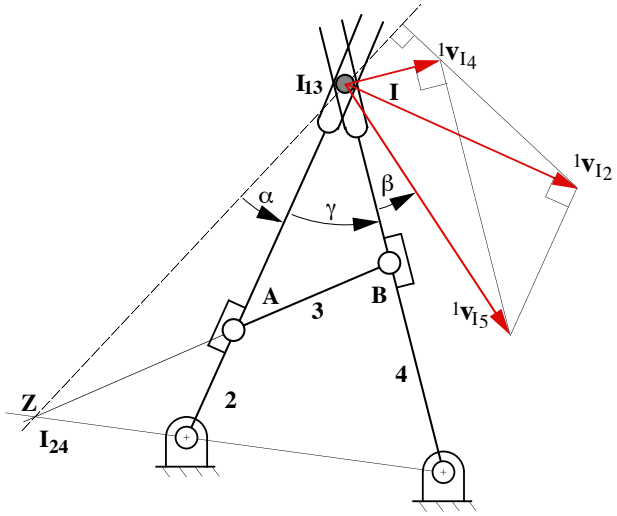


Fig. 4.H.16: Velocity polygon for determining the velocity of the instant center, I_5.

Now,

$^1\boldsymbol{v}_{I_2}$ is perpendicular to line *AI*,

$^1\boldsymbol{v}_{I_4}$ is perpendicular to line *BI*,

$^1\boldsymbol{v}_{I_5/I_2}$ is parallel to line *AI*,

$^1\boldsymbol{v}_{I_5/I_4}$ is parallel to line *BI*,

and

$^1\boldsymbol{v}_{I_2/I_4}$ is perpendicular to the line from I_{24} to I_{13} (Line *ZI*).

Because of the right angles indicated, the ends of vectors $^1\boldsymbol{v}_{I_2}$ and $^1\boldsymbol{v}_{I_4}$ lie on a circle with $^1\boldsymbol{v}_{I_5}$ as the diameter. A detailed representation of the angles involved is shown in Fig. 4.H.17. Because quadrilateral Iacd is inscribed in a circle, two observations can be made from plane geometry:

a) Opposite angles of the quadrilateral are supplementary
b) All angles inscribed by the same chord segment (or equal segments) are equal.

Therefore,

$$\gamma + \beta = \frac{\pi}{2} - \rho$$

$$\gamma + \eta = \frac{\pi}{2} - \alpha$$

Then,

$$\gamma - \frac{\pi}{2} = -(\beta + \rho)$$

$$\gamma - \frac{\pi}{2} = -(\eta + \alpha)$$

and

$$\beta + \rho = \eta + \alpha$$

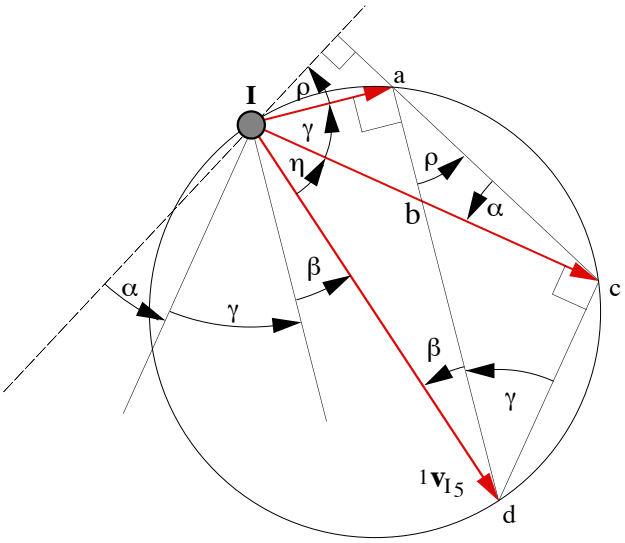


Fig. 4.H.17: Details of the velocity polygon in Fig. 4.H.16.

Also triangles dIc and dac contain a common chord line as a side. Therefore, $\rho = \eta$, which requires that $\alpha = \beta$. Comparing Figs. 4.H.15 and 4.H.16, it is clear that $a = \beta = \theta$. which proves the theorem.

A consequence of the Bobillier theorem is that the direction of the velocity of I_5 is a purely geometric quantity as it should be since the direction of the tangent to the centrodes at the contact point (instant center location) is a purely geometric quantity.

4.H.6.2 First Bobillier Construction

Given the centrode tangent and inflexion circle, construct the center of curvature of the path of any nominated point. The steps are given below, and the construction is shown in Fig. 4.H.18.

Steps

1) Select the circle point *C*
2) Locate point *J* on the opposite end of the diameter of the circle from *I*.
3) Draw line *CI* and construct the normal to *CI* at *I*.
4) Locate point *G* at the intersection of the normal to *CI* at *I* and line *CJ*.
5) Construct the normal to the centrode tangent through point *G*.
6) Locate center point *C** at the intersection of the normal to the centrode tangent through G and line *CI*.

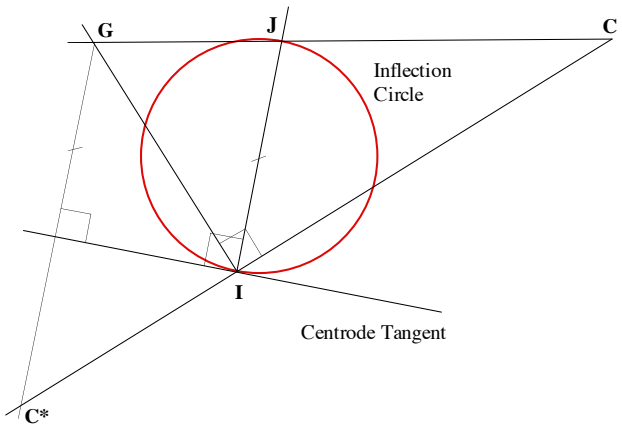


Fig. 4.H.18: First Bobillier construction.

Proof

$$r_{C/I} = IC, \quad \rho = r_{C/C^*}, \quad D = IJ \text{ and } \gamma_C = \angle JIC$$

Triangles IJC and C^*GC are similar so

$$\frac{C^*C}{IC} = \frac{C^*G}{IJ}$$

and $\angle GC^*I = \angle JIC = \gamma_C$.

Therefore

$$C^*G = \frac{C^*I}{\cos\gamma_C} = \frac{C^*C - IC}{\cos\gamma_C} = \frac{\rho - r_{C/I}}{\cos\gamma_C}$$

Also

$$\frac{C^*C}{IC} = \frac{\rho}{r_{C/I}} = \frac{C^*G}{IJ} = \frac{\rho - r_{C/I}}{D\cos\gamma_C}$$

giving

$$\rho D\cos\phi = \rho r_{C/I} - r_{C/I}^2$$

or

$$\rho = \frac{r_{C/I}^2}{r_{C/I} - D\cos\gamma_C}$$

which is the Euler-Savary equation (Eq. H.10).

4.H.6.3 Second Bobillier Construction

Given the inflexion circle and the instantaneous center, find the center points corresponding to two nominated circle points. The steps are given below and the construction is given in Fig. 4.H.19.

Steps

1) Select circle points C and D and draw line CD.

2) Join points C and D to the instantaneous center I to locate points J_C and J_D at the intersections of the

junction lines with the inflexion circle.

3) Join points J_C and J_D to locate point E at the intersection of lines CD and J_CJ_D.

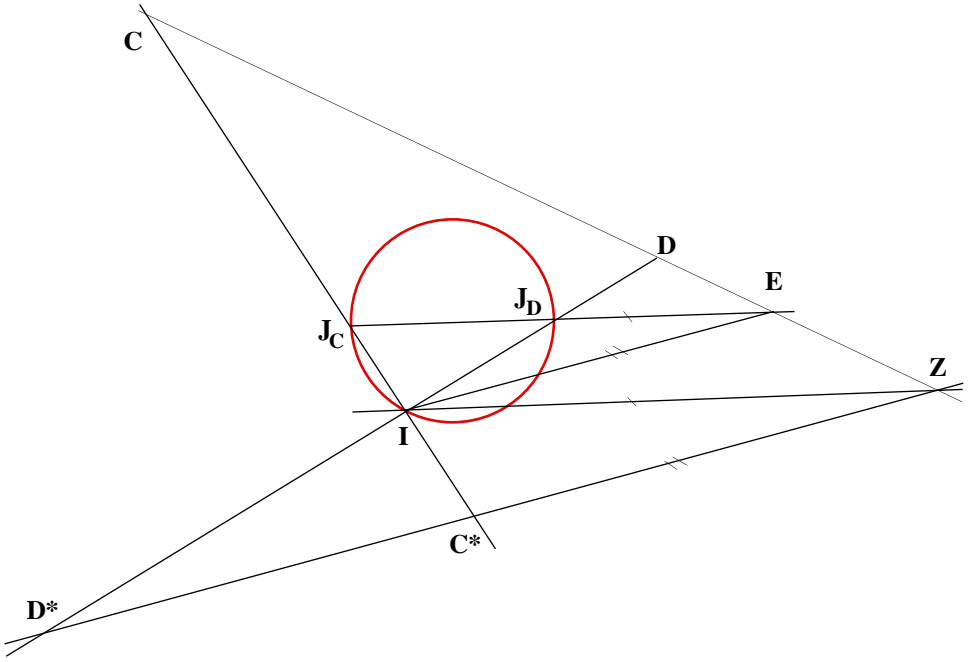


Fig. 4.H.19: The second Bobillier construction.

4) Join I to E.

5) Draw a line parallel to J_CJ_D through I. Its intersection with line CD gives point Z. The collineation axis is line IZ.

6) Draw a line through Z parallel to IE. Its intersections with lines CI and DI give the center points C^* and D^* respectively.

Proof

Triangle ICE is similar to triangle C^*CZ

so $$\frac{IC}{C^*C} = \frac{IE}{C^*Z} \ .$$

Also triangle IJ_CE is similar to triangle C^*IZ

so $$\frac{IE}{C^*Z} = \frac{IJ_C}{C^*I}$$

hence $$\frac{IC}{C^*C} = \frac{IJ_C}{C^*I}$$

Now

$$IC = r_{C/I},\ C^*C = \rho,\ IJ_C = D\cos\gamma_C,\ C^*I = \rho - r$$

so $$\frac{r_{C/I}}{\rho} = \frac{D\cos\gamma_C}{\rho - r_{C/I}} \quad \text{or} \quad \rho = \frac{r_{C/I}^2}{r_{C/I} - D\cos\gamma_C} \ .$$

A similar proof holds for point *D*.

Bobillier's second construction is of greater importance when used in reverse. It then becomes a means of constructing the inflexion circle of a given four-bar linkage. The steps are given below and the construction is shown in Fig. 4.H.20.

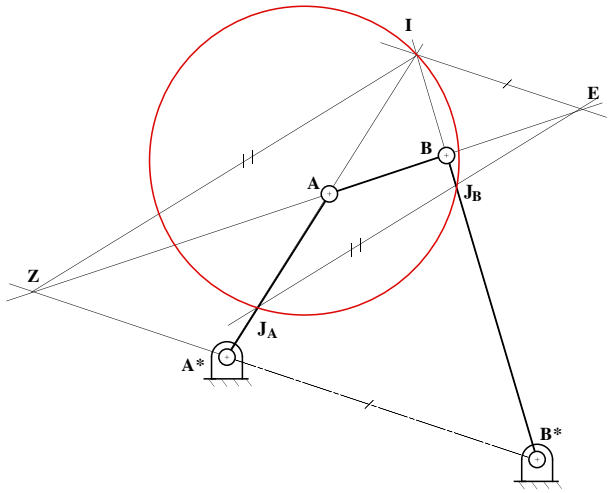


Fig. 4.H.20: The second Bobillier construction used in reverse to find the inflection circle of a given four-bar linkage.

Steps

1) Locate the instantaneous centers *I* and *Z* and draw the collineation axis *IZ*.

2) Draw a line through *I* parallel to line *A*B**. Its intersection with line *AB* is *E*.

3) Draw a line through *E* parallel to *IZ*. Its intersections with lines *A*A* and *B*B* are points J_A, J_B, respectively.

4) Draw a circle through points *I*, J_A, J_B. This is the inflexion circle.

4.H.7 MATLAB Routines for Inflection-Circle Calculations

In this appendix, some of the equations in Table 4.H.1 for the Euler-Savary equation are programmed. Five routines (*inflection1.m, inflection2.m, inflection3.m, inflection_circle.m*, and *inflection_4bar.m*) have been developed. The first three routines are function routines which solve the Euler-Savary equations for different input variables, and the last two routines use the function routines. The routines are discussed in the following.

4.H.7.1 First Inflection Circle Calculation Routine (*inflection1.m*)

The first MATLAB m-file performs the inflection circle calculations when the coordinates of the points *A*, pole *I*, and J_A are known and the coordinates of *A** are to be found. In addition, if γ is known, the coordinates of *J* can be calculated. If the coordinates of *J* are known, γ can be calculated. The initial

statement in the function is

```
function [coords] = inflection1(Ax, Ay, Ix, Iy, JAx, JAy, gamma, JX, JY,
                    flag))
```

The input variables are:

```
Ax    = X coordinate of point
Ay    = Y coordinate of point
Ix    = X coordinate of instant center
Iy    = Y coordinate of instant center
JAx   = X coordinate of JA on inflection circle
JAy   = Y coordinate of JA on inflection circle
gamma = angle, measured CCW about I, from the diameter of the inflection
        circle (ray IJ) to the ray IA
JX    = x coordinate of J
JY    = y coordinate of J
flag  = calculation flag.  If flag=0, gamma is not know.  If flag = 1,
        gamma is known, and the inflection circle and J can be found.  If
        flag = 2, J is known, and the inflection circle and gamma are to be
        found.
```

The results are returned in the vector "coords". The individual components of coords are

```
coords(1) = X coordinate of Astar.
coords(2) = y coordinate of Astar.
coords(3) = x coordinate of J
coords(4) = y coordinate of J
coords(5) = gamma(degrees)
coords(6) = radius of inflection circle
coords(7) = x coordinate of center of inflection circle.
coords(8) = y coordinate of center of inflection circle.
```

Note that the coords(5) - coords(8) are returned as zero if flag = 0.

4.H.7.2 Second Inflection Circle Calculation Routine (*inflection2.m*)

The second MATLAB m-file performs the inflection circle calculations when the coordinates of the points *A,* pole *I*, and *A** are known and the coordinates of J_A are to be found. In addition, if γ is known, the coordinates of *J* can be calculated. If the coordinates of *J* are known, γ can be calculated. The initial statement in the function is

```
function [coords] = inflection2(Ax, Ay, Ix, Iy, Astarx, Astary, gamma, JX, JY,
                    flag)
```

The input variables are:

```
Ax    = X coordinate of point
Ay    = Y coordinate of point
Ix    = X coordinate of instant center
Iy    = Y coordinate of instant center
Astarx= x coordinate of curvature of point path of A.
Astary= y coordinate of curvature of point path of A.
gamma = angle, measured CCW about I, from the diameter of the inflection
        circle (ray IJ) to the ray IA
JX    = x coordinate of J
JY    = y coordinate of J
```

```
flag  = calculation flag.  If flag=0, gamma is not know.  If flag = 1,
        gamma is known, and the inflection circle and J can be found.  If
        flag = 2, J is known, and the inflection circle and gamma are to be
        found.
```

The results are returned in the vector "coords". The individual components of coords are

```
coords(1) = X coordinate of JA.
coords(2) = y coordinate of JA.
coords(3) = x coordinate of J
coords(4) = y coordinate of J
coords(5) = gamma(degrees)
coords(6) = radius of inflection circle
coords(7) = x coordinate of center of inflection circle.
coords(8) = y coordinate of center of inflection circle.
```

Note that the coords(5) - coords(8) are returned as zero if flag = 0.

4.H.7.3 Third Inflection Circle Calculation Routine (*inflection3.m*)

The third MATLAB m-file performs the inflection circle calculations when the coordinates of the points *A* and of the pole *I*, and the inflection circle are known, and the coordinates of *A** are to be found. Alternatively, if γ is known, the coordinates of *J* can be calculated. If the coordinates of J are known, γ can be calculated. This routine calls *inflection1.m* and *inflection2.m*. The initial statement in the function is

```
function [coords] = inflection3(Ax, Ay, Ix, Iy, Astarx, Astary, Cx, Cy,
                    Jx, Jy, gamma, ptflag, icflag)
```

The input variables are:

```
Ax     = X coordinate of point
Ay     = Y coordinate of point
Ix     = X coordinate of instant center
Iy     = Y coordinate of instant center
Astarx = x coordinate of curvature of point path of A.
Astary = y coordinate of curvature of point path of A.
Cx     = x coordinate of center of inflection circle
Cy     = y coordinate of center of inflection circle
Jx     = x coordinate of J
Jy     = y coordinate of J
gamma  = angle, measured CCW about I, from the diameter of the inflection
         circle (ray IJ) to the ray IA
ptflag = point flag.  If ptflag = 0, the point (A) is known and center of
         the path is to be found.  If ptflag = 1, the center of the path
         (Astar) is known, and the point (A) is to be found.
icflag = inflection circle flag.  If flag=0, Cx and Cy are know.  If flag =
         1, Jx and Jy are known, and the inflection circle and JA can be
         found.
```

The results are returned in the vector "coords". The individual components of coords are

```
coords(1) = X coordinate of JA.
coords(2) = y coordinate of JA.
coords(3) = x coordinate of J
coords(4) = y coordinate of J
coords(5) = gamma(degrees)
coords(6) = radius of inflection circle
```

```
coords(7) = x coordinate of center of inflection circle.
coords(8) = y coordinate of center of inflection circle.
coords(9) = x coordinate of point A
coords(10)= y coordinate of point A
coords(11)= x coordinate of point Astar
coords(12)= y coordinate of point Astar
```

Note that the coords(5) - coords(8) returned as zero if flag = 0.

4.H.7.4 Inflection Circle Routine (*inflection_circle.m*)

This MATLAB routine graphically displays the solutions of the Euler-Savary equation. The inflection circle is displayed, and the user can select points (*A*) for which the center point (*A**) is to be determined. The user can select up to 10 points (*A*'s) for which the corresponding center points (A*'s) are to be found. The input can be through the mouse or keyboard. This routine uses *inflection3.m*. The inputs to the routine are

```
Cx     = x coordinate of center of inflection circle
Cy     = y coordinate of center of inflection circle
theta  = arc distance on inflection circle which is to be shown at A to
         approximate the path of A on the ground link.
Ix     = X coordinate of instant center
Iy     = Y coordinate of instant center
sizefact = size factor for plot.  The window size is (1+2sizefact)D where D
           is the diameter of the inflection circle
Ax     = X coordinate of point
Ay     = Y coordinate of point
```

4.H.7.5 Sample run using *inflection_circle.m*

A copy of the input screen is given in Table 4.H.2 and the plot is displayed in Fig. 4.H.21. Three points have been selected for evaluation, and the input was through the mouse.

Table 4.H.2: Input and output corresponding to sample analysis

```
      Inflection Circle Test Program
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (inflectioncirio.dat): manual.dat
Enter x coordinate of center of inflection circle: 1
Enter y coordinate of center of inflection circle: 1
Enter arc from inflection circle for indicating path (deg): 30
Enter x coordinate of instant center: 0
Enter y coordinate of instant center: 0
Enter size factor for plot: 1

Possible modes of inputting data:
  1 - mouse
  2 - keyboard

Designate mode of input [1] 1

  Use mouse to locate point

reenter point? y/n [y]: y

  Use mouse to locate point
```

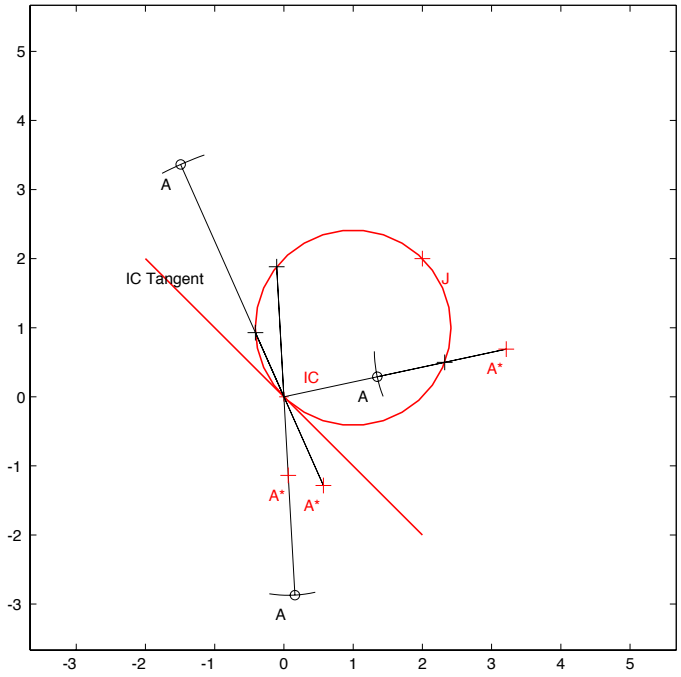


Fig. 4.H.21: Sample run of inflection circle routine.

4.H.7.6 Inflection Circle Routine for Four-Bar Linkage (*inflection_4bar.m*)

This MATLAB routine graphically displays the solutions of the Euler-Savary equation for the coupler of a four-bar linkage. The inflection circle is displayed, and the user can select points (A) for which the center point (A*) is to be determined. The user can select up to 10 points (A's) for which the corresponding center points (A*'s) are to be found. The input can be through the mouse or keyboard. This routine uses *inflection_circle.m*. The inputs to the routine are

```
r1     = length of the frame
r2     = length of the crank
r3     = length of the coupler
r4     = length of the rocker
Q1     = frame angle in degrees
Q2     = crank angle in degrees
theta  = arc distance on inflection circle which is to be shown at A to
         approximate the path of A on the ground link.
mode   = assembly mode for linkage
```

4.H.7.6.1 Sample run using *inflection_4bar.m*

A copy of the input screen is given in Table 4.H.3 and the plot is displayed in Fig. 4.H.22. Three points have been selected for evaluation, and the input was through the mouse.

Table 4.H.3: Input and output corresponding to sample analysis using *inflection_4bar.m*

```
 Inflection Circle Program for Four-Bar Linkage

Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (inflection4bario.dat): manual.dat
Enter the frame length (cm) [6.928]: 6.928
Enter the crank length (cm) [4]: 4
Enter the coupler length (cm) [4]: 4
Enter the rocker length (cm)[4*sqrt(3)]:
Enter the frame angle (deg) [200]: 120
Enter the crank angle (deg) [90]: 90
Enter arc from inflection circle for indicating path (deg) [30]: 30

Enter assembly mode [1]: 1
I am working ...

Possible modes of inputting data:
  1 - mouse
  2 - keyboard

Designate mode of input [1] 1
  Use mouse to locate point

Reenter point? y/n [y]: y
  Use mouse to locate point

Reenter point? y/n [y]: y
  Use mouse to locate point
Reenter point? y/n [y]: n
```

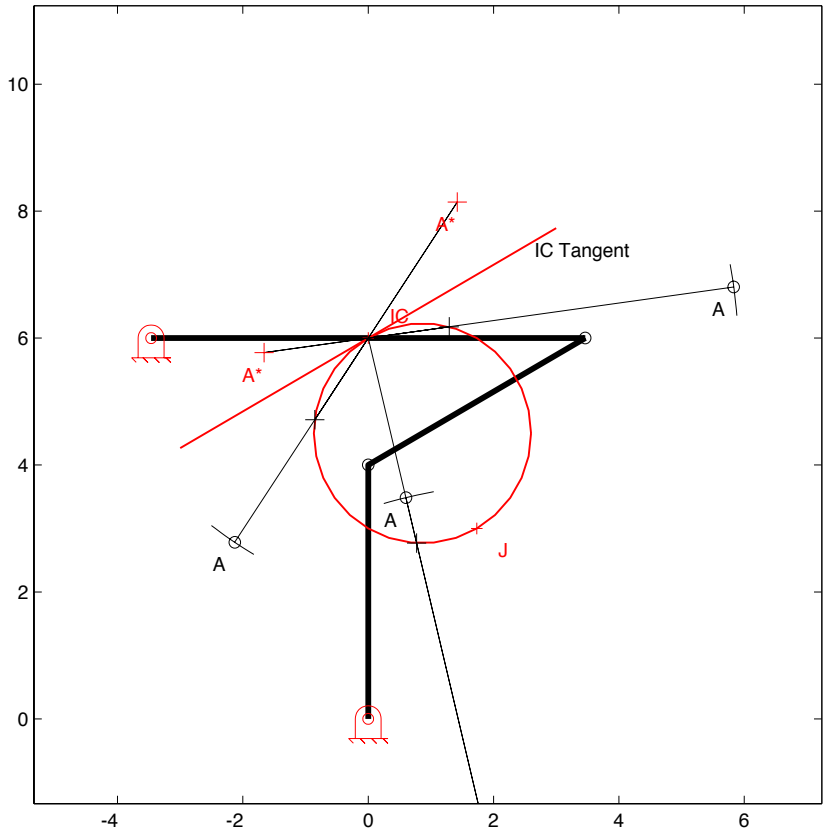


Fig. 4.H.22: Sample run of inflection circle routine for 4-bar linkage.

5.0 Programs for Chapter 5

(No programs have been written for Chapter 5)

6.0 Programs for Chapter 6

MATLAB Programs for Cam Analysis

Chapter 6 includes 4 appendices which contain a description of the MATLAB function routines developed to generate the cam profiles for the four main types of cam-follower systems discussed in this Chapter 6. The source code for the functions is included on the disk included with this book.

6.A MATLAB Procedure for Cam Design for Axial Cylindrical-Faced Follower

6.A.1 Overview

In this section, the equations given in Table 6.8 are programmed in m-files for MATLAB, and Example 6.7 is solved to illustrate the use of the program. To use the program, a function routine (*follower.m*) must be written to provide $y(\theta)$ in an array (θ, y). The routine *rf_cam.m* uses this routine to determine the cam profile and to plot the results. Other information which must be known are the base circle radius (r_b), the roller radius (r_0), the offset distance δ, the cam angle increment for conducting the analysis, and the direction of rotation [CCW +, CW -]. The program first plots the cam profile and then animates the cam follower system as a function of θ.

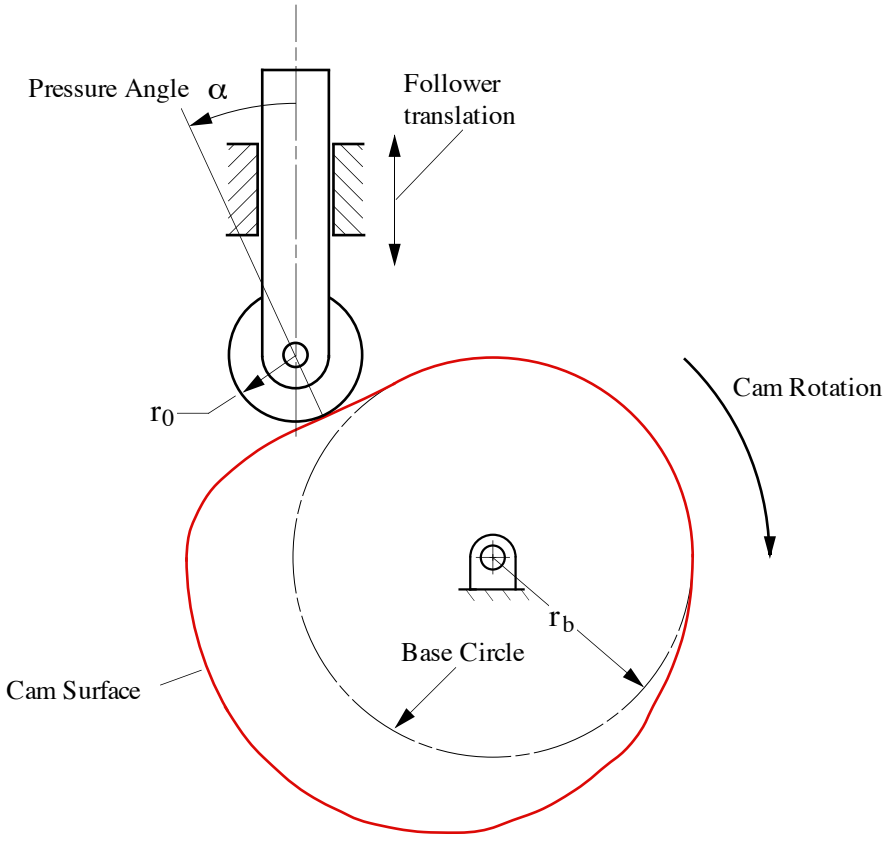


Fig. 6.A.1: Basic cam with axial roller follower.

In addition to using *follower.m*, the routine also uses four other MATLAB routines. These are:

axisadjust.m, pole.m, circle.m, *bushing.m*, and *frameline.m*. All of these have already been discussed. The results for Example 6.7 are given following the program description.

6.A.2 Matlab Routines for Roller Follower Cam System (*rf_cam.m*)

This program determines the cam profile for a translating roller follower. The input variables for the routine are:

```
rb        = base circle radius [2]
r0        = radius of cylindrical or roller follower [0.5]
d         = follower offset [1]
rise      = follower rise [2]
direction = cam rotation direction (CW(-), CCW(+)) [-]
ainc      = cam angle increment for design (deg) [10]
ncycle    = number of animation cycles [4]
```

The numbers in the brackets are the default values.

6.A.3 Follower Routine (*follower.m*)

This function determines the follower displacement and derivatives for a full rotation cam. The routine is set up for the displacement schedule in Example 6.7. The initial statement in the function is:

```
function [f]= follower(tt,rise)
```

The input values are:

```
tt     = cam angle (deg)
rise   = maximum follower displacement
```

T) is the displacement, f(2) is the derivative of the displacement with respect to theta, and f(3) is the second derivative with respect to theta. The function routine for Exammple 6.8 is given in the following.

```
function [f]= follower(tt,rise)

% This function determines the follower displacement and derivatives
% for a full rotation cam.  The routine is set up for the displacement
% schedule in Examples 6.7 and 6.8

% The input values are:

%theta   = cam angle (deg)
%rise    = maximum follower displacement

% The results are returned in the variable f where f(1) is the
% displacement, f(2) is the derivative of the displacement with
% respect to theta, and f(3) is the second derivative with respect
% to theta.

% find the correct interval.

fact=pi/180;
theta=tt*fact;
if theta < pi/2
      f(1)=0;
      f(2)=0;
      f(3)=0;
end
```

```
if theta >= pi/2 & theta < pi
      f(1)=(rise/(pi/2))*((theta-pi/2)-(sin(4*(theta-pi/2)))/4);
      f(2)=(rise/(pi/2))*(1-cos(4*(theta-pi/2)));
      f(3)=(8*rise/pi)*sin(4*(theta-pi/2));
end
if theta>=pi & theta< 4*pi/3
      f(1)=rise;
      f(2)=0;
      f(3)=0;
end
if theta >= 4*pi/3
      f(1)=(rise/2)*(1+cos(1.5*(theta-4*pi/3)));
      f(2)=-(3*rise/4)*(sin(1.5*(theta-4*pi/3)));
      f(3)=-(9*rise/8)*(cos(1.5*(theta-4*pi/3)));
end
```

6.A.4 Results from Sample Run of *rf_cam* (Example 6.7)

The sample run solves the problem in Example 6.7. The input data is given in Table 6.A.1 and the graphical results are given in Figs. 6.A.2 and 6.A.4. The follower displacement is given in *follower.m.*

Table 6.A.1: Input Data and Numerical Results from *rf_cam.m*

```
          Cam Synthesis for Axial Roller Follower

    Enter 1 for file input and 2 for interactive input [1]: 2
    Enter input file name (rf_camio.dat): manual.dat
    Enter base circle radius [2]:  4
    Enter radius of cylindrical or roller follower [0.5]: 1
    Enter follower offset [1]:  0.5
    Enter follower rise [2]:  2
    Enter cam rotation direction (CW(-), CCW(+)) [-]: -
    Enter cam angle increment for design (deg) [10]:  0.5
    Enter number of animation cycles [4]:  3
    Reinter input variables? y/n [n]: n

       Input Variables
     Base Radius   Follower Radius   Follower Offset    rise
        4.000           1.000             0.500        2.000

     *** Computing the cam profile ***
    Repeat animation? y/n [y]: »
```

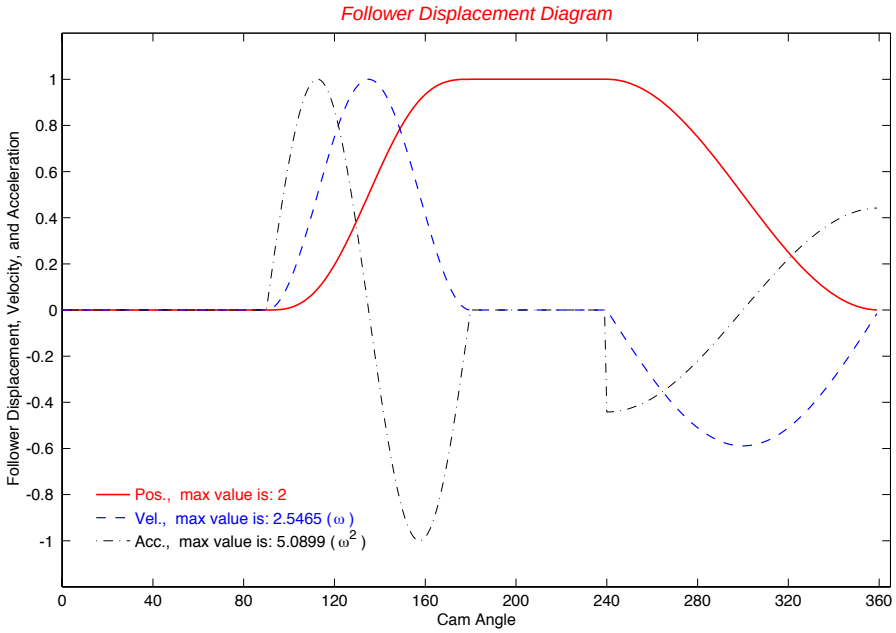


Fig. 6.A.2: Follower displacement schedule for Example 6.7.

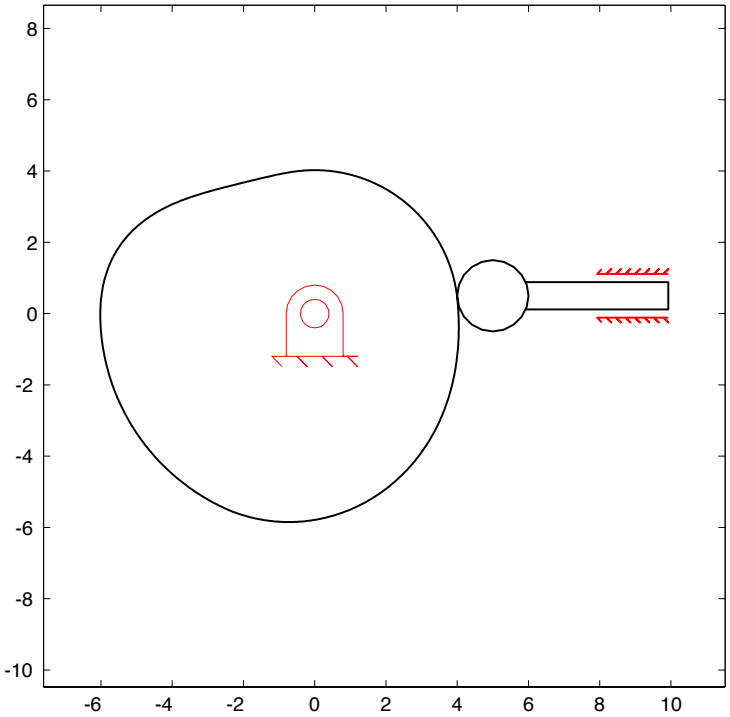


Fig. 6.A.3: Basic cam design.

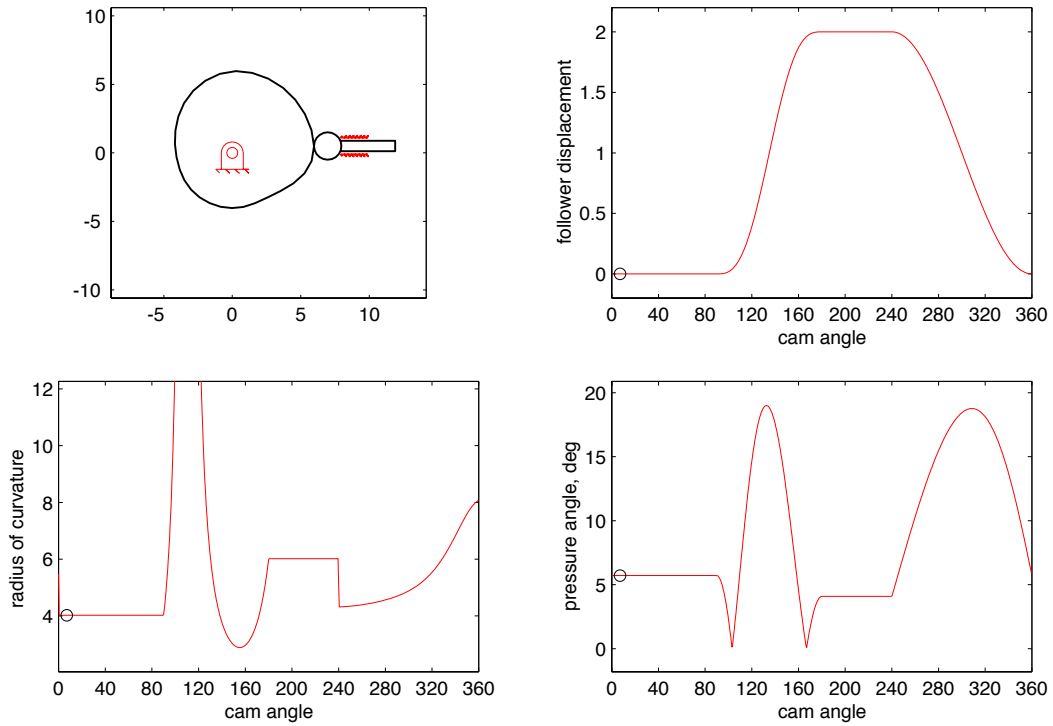


Fig. 6.A.4: Cam geometry, follower displacement, radius of curvature, and pressure angle plots.

6.B MATLAB Procedure for Cam Design for Axial Flat-Faced Follower

6.B.1 Overview

In this appendix, the equations given in Table 6.8 are programmed in m-files for MATLAB, and Example 6.8 is solved to illustrate the use of the program. To use the program, a function routine (*follower.m*) must be written to provide $y(\theta)$ in an array (θ,y). This routine is the same as that described in Appendix 6.A. The routine *ff_cam.m* uses this routine to determine the cam profile and plot the results. Other information which must be known are the base circle radius (r_b), the maximum and minimum values for t (see Fig. 6.B.1), and the direction of rotation of the cam. The program first plots the cam profile and then animates the cam follower system as a function of θ.

In addition to using *follower.m*, the routine also uses four other MATLAB routines. These are: *axisadjust.m, rect.m, circle.m*, *bushing.m*, *rectangle.m*, and *frameline.m*. All of these have already been discussed. All of the routines are self explanatory and the logic is described in the comment statements. The results for Example 6.8 are given following the description of *ff_cam.m*.

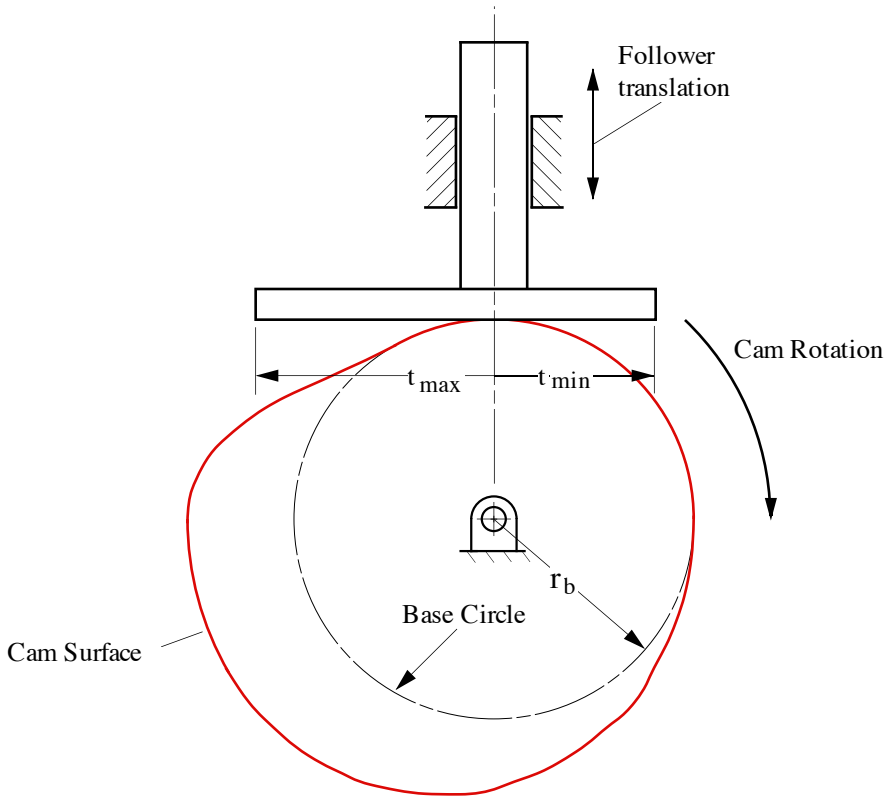


Fig. 6.B.1: Basic cam with axial flat-faced follower.

6.B.2 Matlab Design Routines for Flat-Faced Follower Cam System (*ff_cam.m*)

This program determines the cam profile for a translating flat-faced follower. The input variables for the routine are:

```
rb        = base circle radius [3.2]
rise      = follower rise [2]
tmin      = distance from follower centerline to leftmost end of follower
            face [1.6]
tmax      = distance from follower centerline to rightmost end of follower
            face [2.6]
direction = cam rotation direction (CW(-), CCW(+)) [-]
ainc      = cam angle increment for design (deg) [10]
ncycle    = number of animation cycles [4]
```

The numbers in the brackets are the default values.

6.B.3 Results from Sample Run of *ff_cam* (Example 6.8)

The sample run solves the problem in Example 6.8. The input data is given in Table 6.B.1 and the graphical results are given in Figs. 6.B.2 and 6.B.4. The follower displacement is given in *follower.m.* discussed in Section 6.A.

Table 6.B.1: Input Data and Numerical Results from *ff_cam.m*

```
      Cam Synthesis for Axial Flat-Faced Follower

Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (ff_camio.dat): manual.dat
Enter base circle radius [3.2]:  4
Enter follower rise [2]:  1
Enter distance from follower centerline to bottom of face [1.6]:  1.6
Enter distance from follower centerline to top of face [2.6]:  2.6
Enter cam rotation direction [CW(-), CCW(+)] [-]: -
Enter cam angle increment for design (deg) [10]:  0.5
Enter number of animation cycles [4]:  3
Reinter input variables? y/n [n]: n

             Input Variables
  Base Radius    tmin       tmax        rise
     4.000       1.600      2.600      1.000

 *** Computing the cam profile ***
Repeat animation? y/n [y]:
Change for animation speed? +/-/0 [-]:
Repeat animation? y/n [y]: n
```

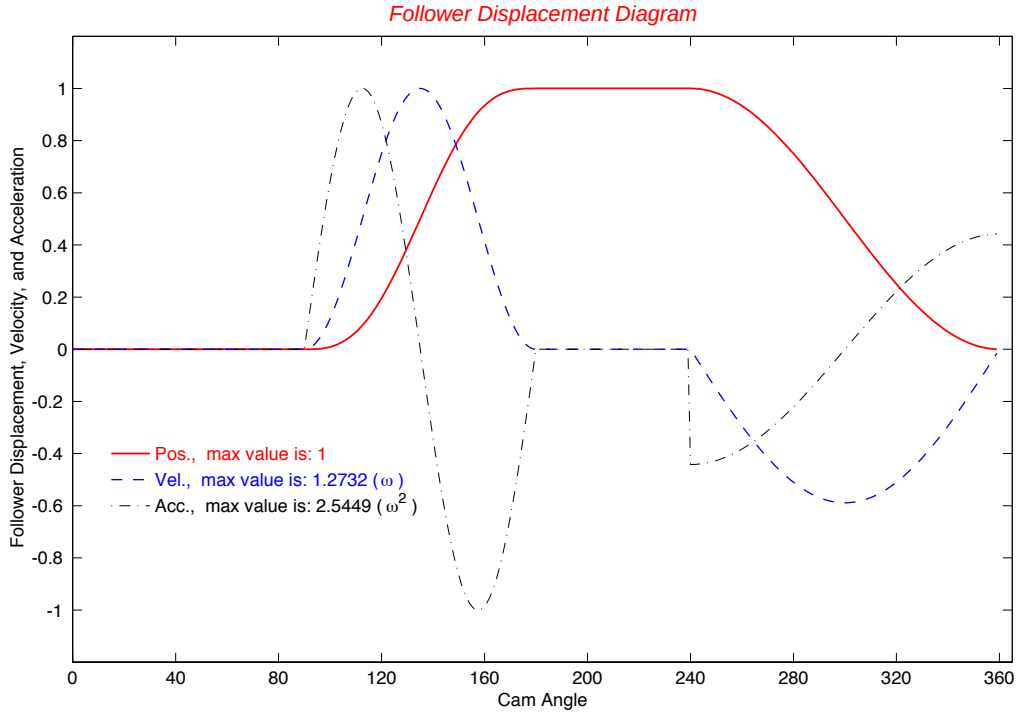


Fig. 6.B.2: Follower displacement schedule for Example 6.8.

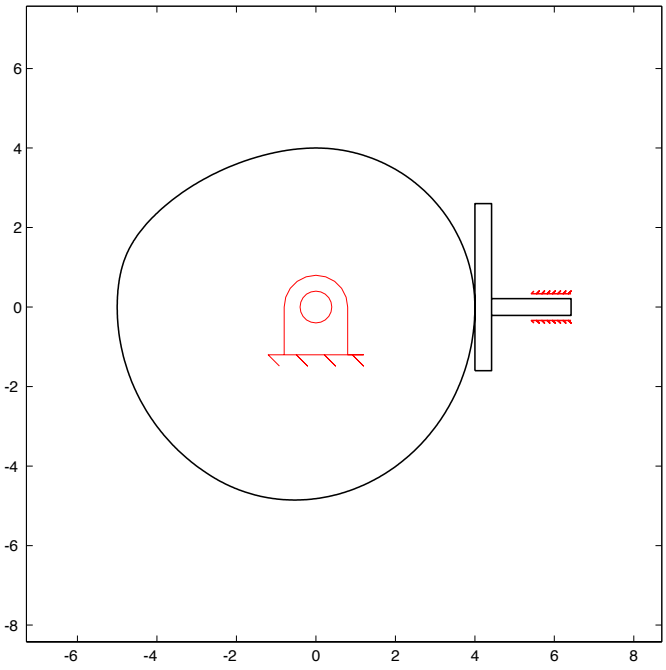


Fig. 6.B.3: Basic cam design.

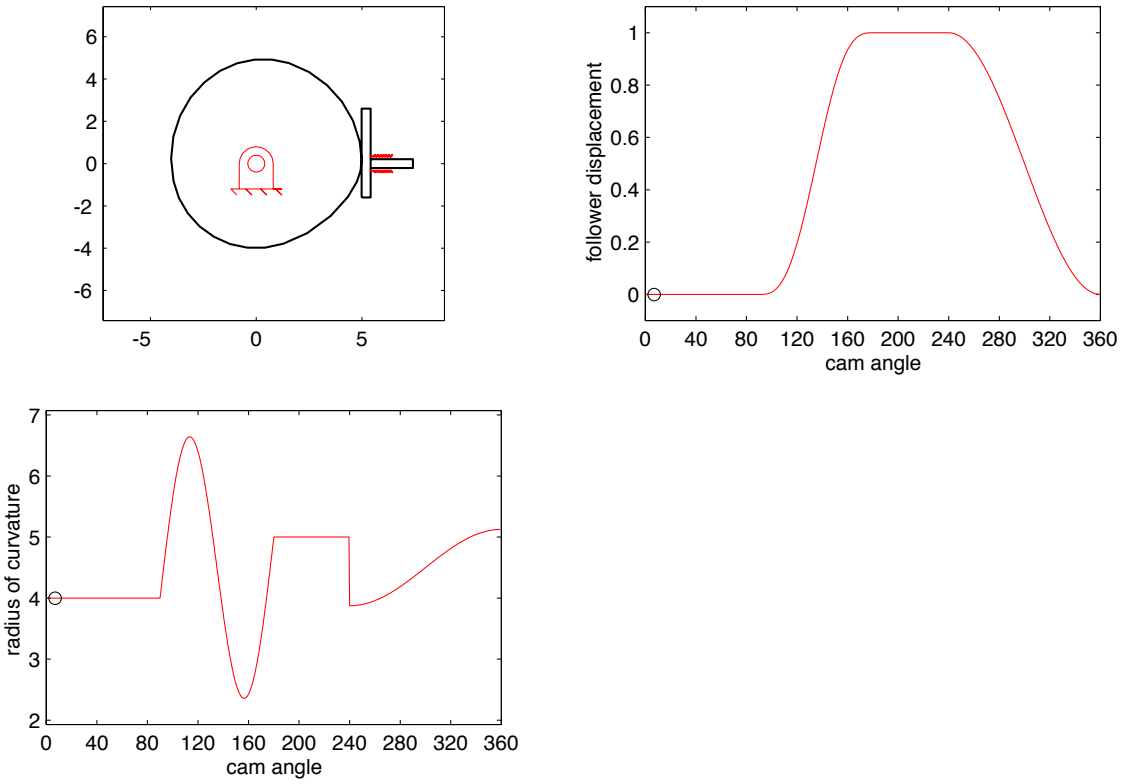


Fig. 6.B.4: Cam geometry, radius of curvature, and follower displacement plots.

6.C MATLAB Procedure for Cam Design for Oscillating, Cylindrical-Faced Follower

6.C.1 Overview

In this appendix, the equations given in Table 6.10 are programmed in m-files for MATLAB, and Example 6.9 is solved to illustrate the use of the program. To use the program, a function routine (o_*follower.m*) must be written to provide $\phi(\theta)$ in an array (θ,ϕ). The routine o*rf_cam.m* uses this routine to determine the cam profile and plot the results. Other information which must be known are the base circle radius (r_b), the roller radius (r_0), the distance between the cam and follower pivots (r_1), and distance from the follower pivot and the center of the follower (r_3). The program first plots the cam profile and then animates the cam follower system as a function of θ.

In addition to using o_*follower.m*, the routine uses four other MATLAB routines. These are: *axisadjust.m, pole.m, circle.m*, and *bushing.m*. All of these were discussed previously. The results for Example 6.9 are given following the descriptions of o_*follower.m* and o*rf_cam.m*.

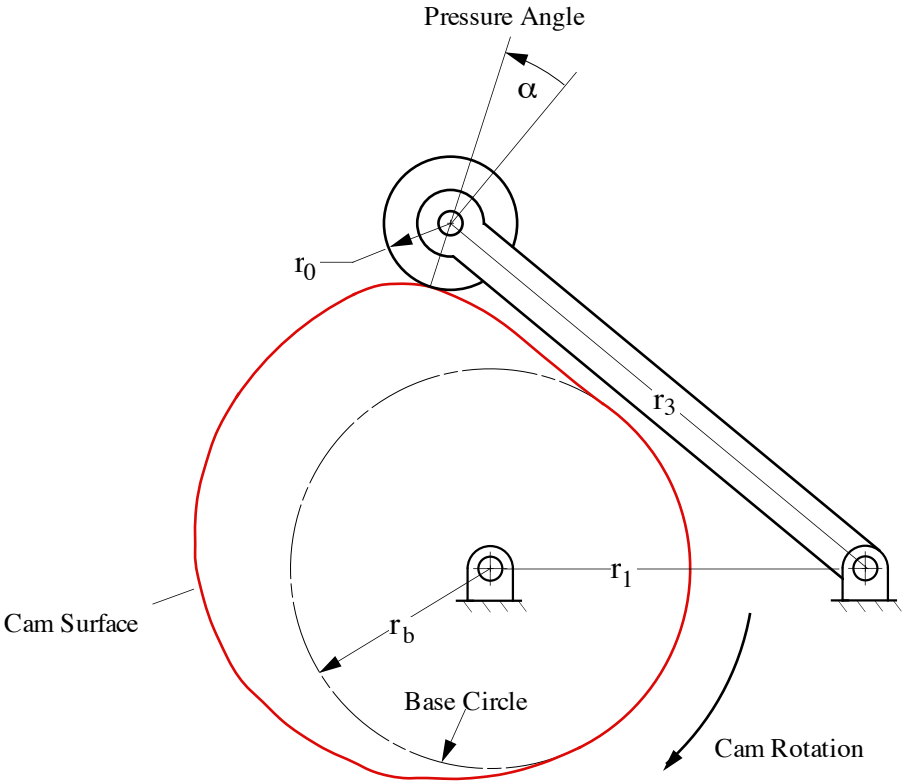


Fig. 6.C.1: Basic cam with axial roller follower.

6.C.2 Matlab Routines for Oscillating, Cylindrical-Faced Follower Cam System (*orf_cam.m*)

This program determines the cam profile for an oscillating, cylindrical-faced follower. The input variables for the routine are:

```
rb        = base circle radius [2]
r0        = radius of cylindrical or roller follower [1]
r1        = distance between fixed pivots [3+rb]
r3        = follower length [r3]
d         = follower offset [1]
rised      = follower rise (deg) [30]
direction = cam rotation direction (CW(-), CCW(+)) [-]
ainc      = cam angle increment for design (deg) [10]
ncycle    = number of animation cycles [4]
```

The numbers in the brackets are the default values.

6.C.3 Follower Routine (*o_follower.m*)

This function determines the follower displacement and derivatives for a full rotation cam. The routine is set up for the displacement schedule in Example 6.9. The initial statement in the function is:

```
function [f]= o_follower(tt,rise)
```

The input values are:

```
tt     = cam angle (deg)
rise   = maximum follower displacement (rad)
```

T) is the displacement, f(2) is the derivative of the displacement with respect to theta, and f(3) is the second derivative with respect to theta.

6.C.4 Results from Sample Run of *orf_cam* (Example 6.9)

The sample run solves the problem in Example 6.9. The input data is given in Table 6.C.1 and the graphical results are given in Figs. 6.C.2 and 6.C.3. The follower displacement is computed in *o_follower.m*.

Table 6.C.1: Input Data and Numerical Results from orf_cam.m

```
     Cam Synthesis for Oscillating Cylindrical-Faced Follower

Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (orf_camio.dat): manual.dat
Enter base circle radius [2]:  2
Enter radius of cylindrical or roller follower [1]: 1
Enter distance between fixed pivots [3*rb]:  6
Enter follower length [sqrt(r1^2-(rb+r0)^2)]:  5.196
Enter follower rise (deg) [30]:  30
Enter cam rotation direction (CW(-), CCW(+)) [+]: +
Enter angle increment for design (deg) [10]:  0.5
Enter number of animation cycles [4]:  3
Reinter input variables? y/n [n]: n

                        Input Variables
  Base Rad.   Pivot Dist.   Follower Rad.   Follower Length    rise
    2.000       6.000           1.000           5.196         30.000

 *** Computing the cam profile ***
```

```
Repeat animation? y/n [y]: n
```

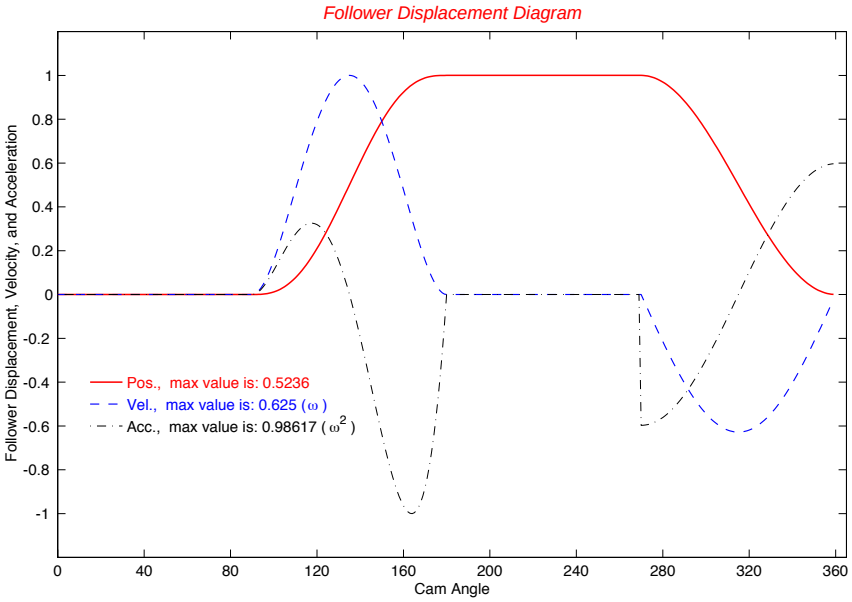


Fig. 6.C.2: Follower displacement schedule for Example 6.9.

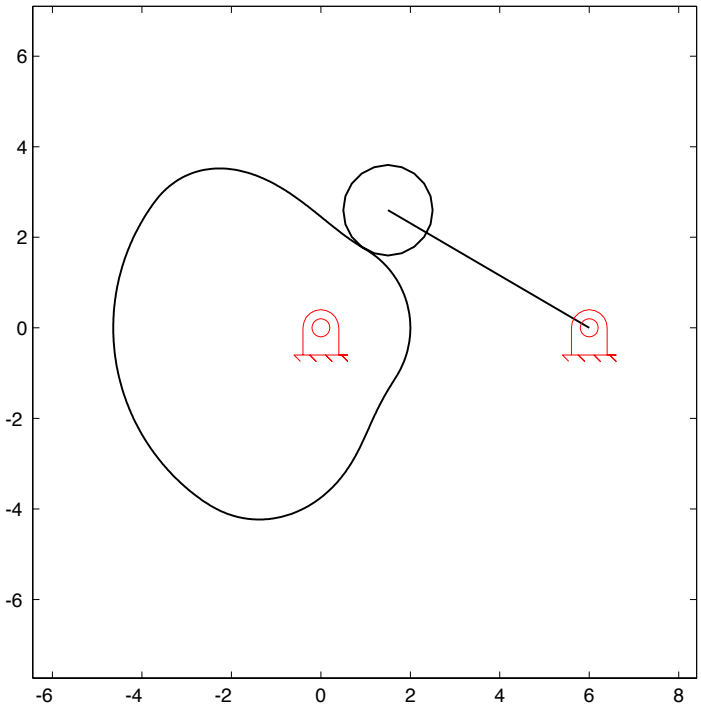


Fig. 6.C.3: Basic cam design.

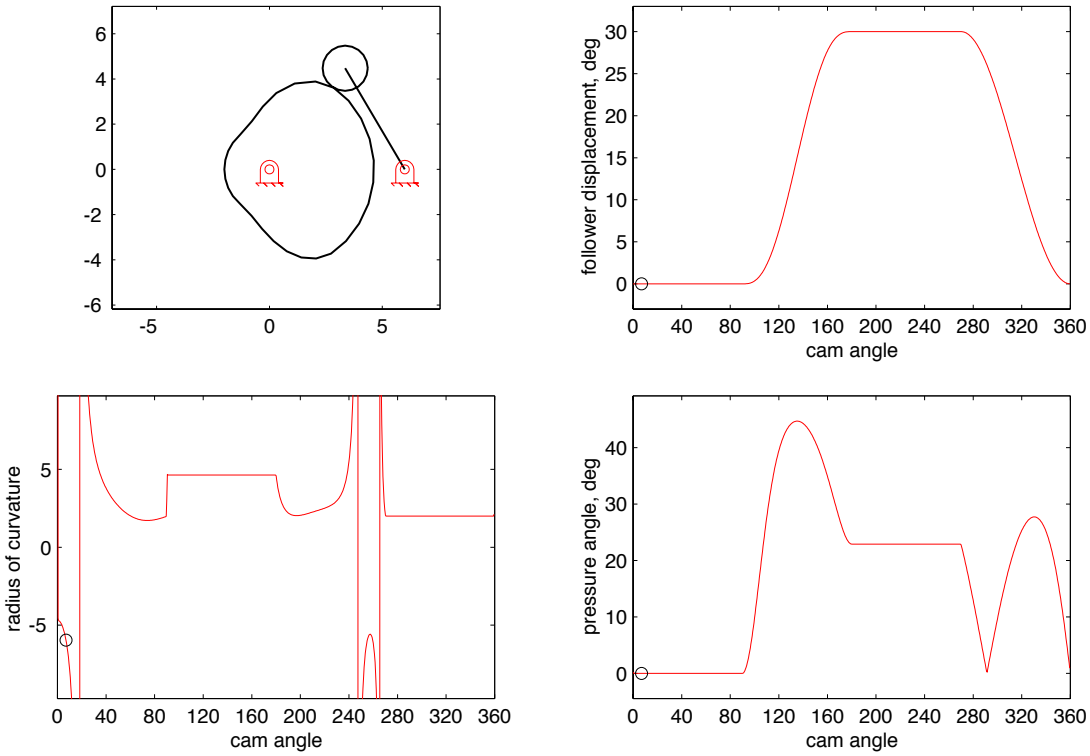


Fig. 6.C.4: Cam geometry, follower displacement, radius of curvature, and pressure angle plots.

6.D MATLAB Procedure for Cam Design for Oscillating, Flat-Faced Follower

6.D.1 Overview

In this appendix, the equations given in Tables 6.12 and 6.13 are programmed in m-files for MATLAB, and Example 6.10 is solved to illustrate the use of the program. To use the program, a function routine (o_*follower2.m*) similar to *o_follower.m* described in Appendix 6.C must be written to provide $\phi(\theta)$ in an array (θ,ϕ). The routine o*ff_cam.m* uses this routine to determine the cam profile and plot the results. Other information which must be known are the base circle radius (r_b), the follower offset (d), and the distance between the cam and follower pivots (r_1). The program first plots the cam profile and then animates the cam follower system as a function of θ.

In addition to using o_*follower2.m*, the routine also uses four other MATLAB routines. These are: *axisadjust.m, pole.m, circle.m*, and *bushing.m*. All of these have already been discussed previously. The results for Example 6.10 are given in the following.

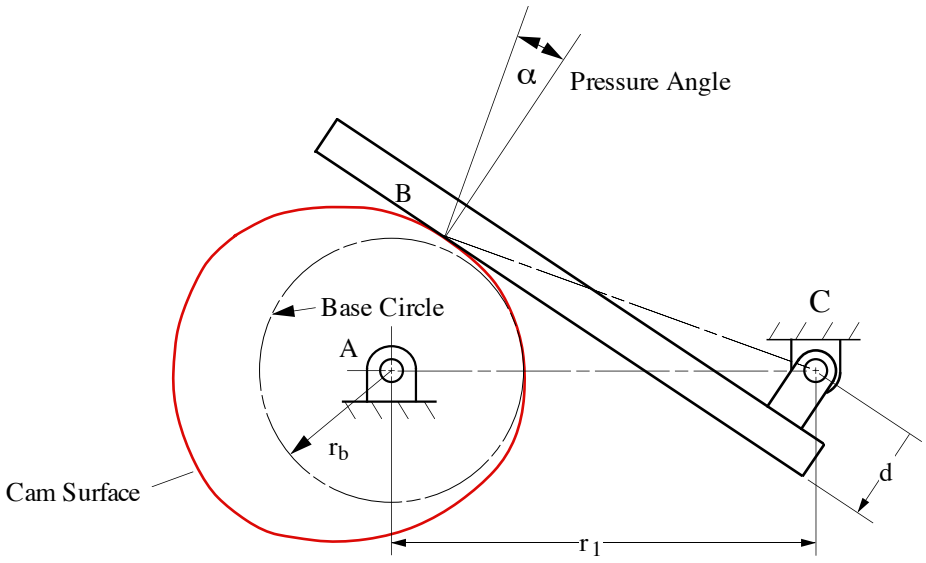


Fig. 6.D.1: Basic cam with axial flat-faced follower.

6.D.2 Matlab Design Routines for an Oscillating, Flat-Faced Follower Cam System (*off_cam.m*)

This program determines the cam profile for an oscillating, flat-faced follower. The input variables for the routine are:

```
rb        = base circle radius [2]
r1        = distance between fixed pivots [6]
r3        = length of follower face [1.5*r1]
d         = follower offset [0.5]
rised     = follower rise in degrees [15]
direction = cam rotation direction (CW(-), CCW(+)) [-]
ainc      = cam angle increment for design (deg) [10]
ncycle    = number of animation cycles [4]
```

6.D.3 Follower Routine (o_follower2.m)

This function determines the follower displacement and derivatives for a full rotation cam. The routine is set up for the displacement schedule in Example 6.10. The initial statement in the function is:

```
function [f]= o_follower2(tt,rise)
```

The input values are:

```
tt     = cam angle (deg)
rise   = maximum follower displacement (rad)
```

T) is the displacement, f(2) is the derivative of the displacement with respect to theta, and f(3) is the second derivative with respect to theta.

6.D.4 Results from Sample Run of off_cam (Example 6.10)

The sample run solves the problem in Example 6.10. The input data is given in Table 6.D.1 and the graphical results are given in Figs. 6.D.2 and 6.D.3. The follower displacement is computed in *o_follower2.m*.

Table 6.D.1: Input Data and Numerical Results from *off_cam.m*

```
      Cam Synthesis for Oscillating Flat-Faced Follower

Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (off_camio.dat): manual.dat
Enter base circle radius [2]:  2
Enter distance between fixed pivots [6]:  6
Length of follower face [1.5*r1]:  9
Enter follower offset [0.5]:  0.5
Enter follower rise (deg) [15]:  20
Enter cam rotation direction (CW(-), CCW(+)) [-]: -
Enter angle increment for design (deg) [10]:  0.5
Enter number of animation cycles [4]:  3
Reinter input variables? y/n [n]: n

                        Input Variables
  Base Rad.   Pivot Dist.  Follower offset  Follower Length     rise
    2.000       6.000           0.500           9.000        20.000

 *** Computing the cam profile ***
Repeat animation? y/n [y]:
```

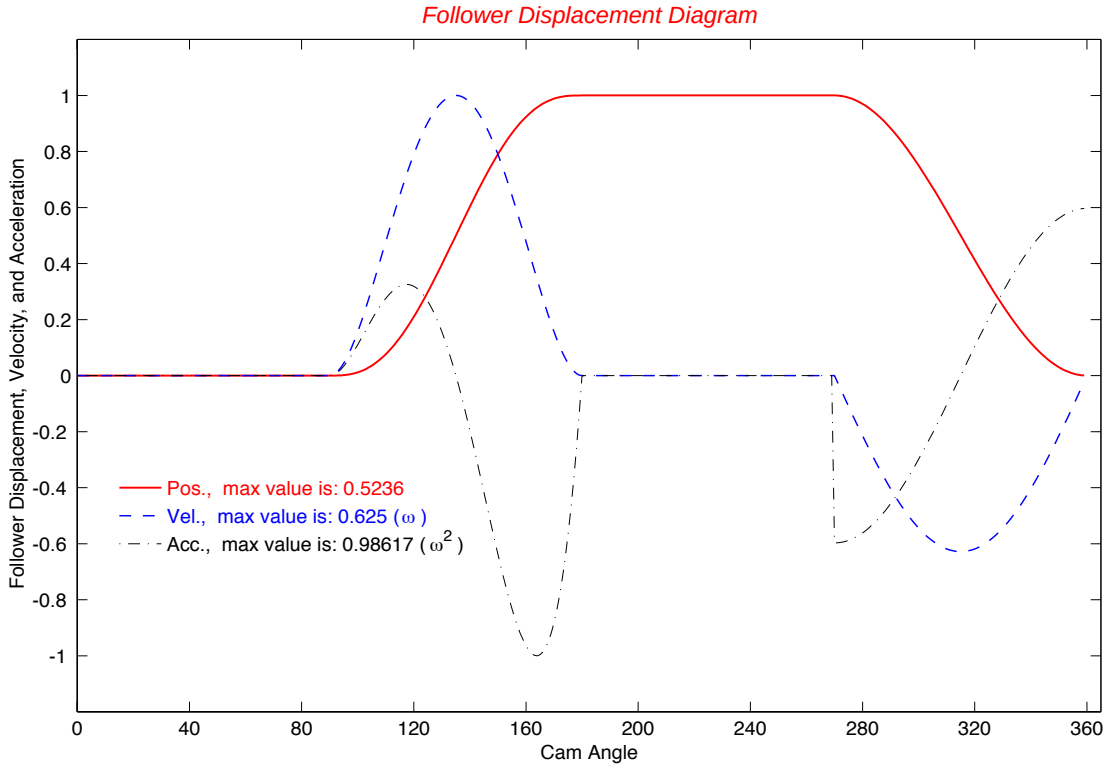


Fig. 6.D.2: Follower displacement schedule for Example 6.10.

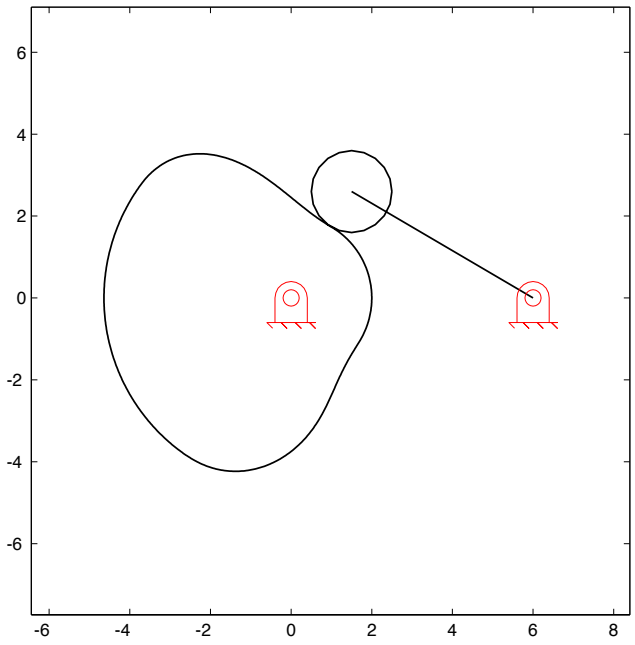


Fig. 6.D.3: Basic cam design.

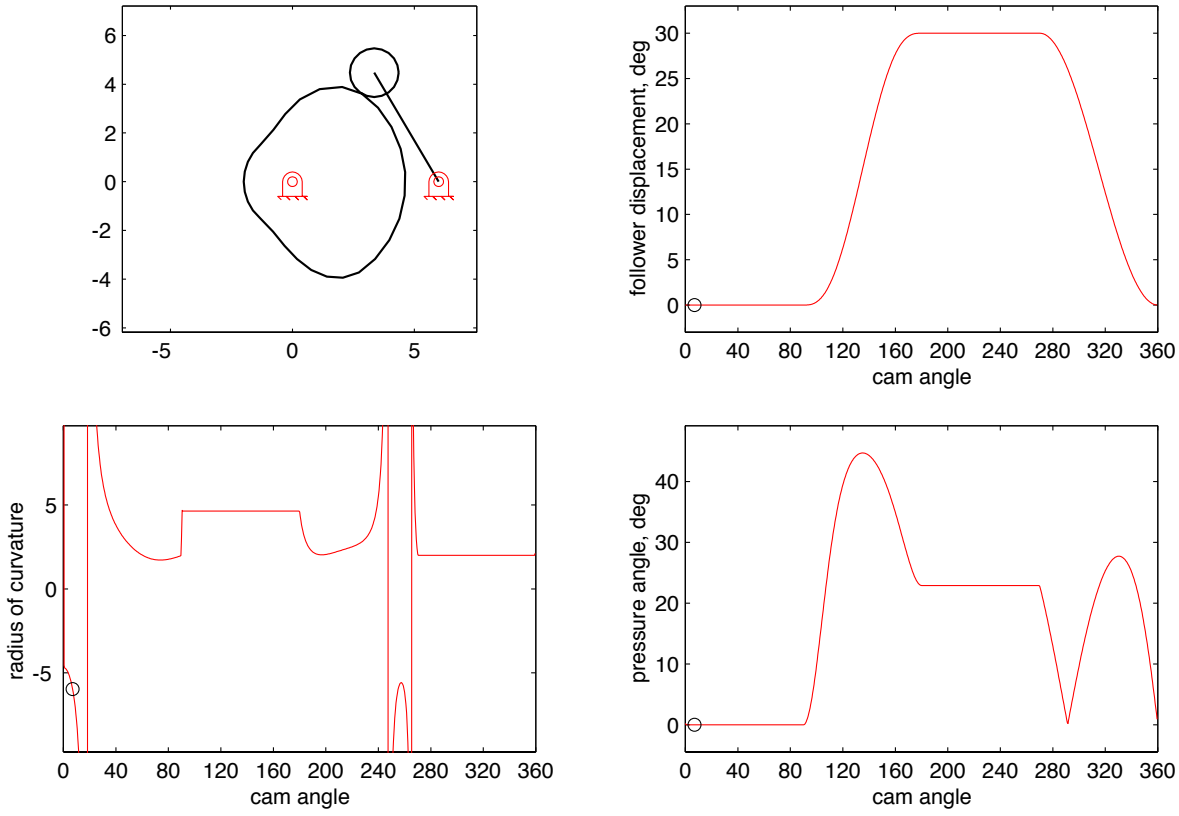


Fig. 6.D.4: Cam geometry, follower displacement, radius of curvature, and pressure angle plots.

7.0 Programs for Chapter 7

(No programs have been written for Chapter 7)

8.0 Programs for Chapter 8: Gear Drawing and Analysis

8.A MATLAB Program for Finding the Inverse Involute

In this appendix, the inverse involute function is determined. That is, given y = inv(x), the function finds x. This can be done in a variety of ways; however, the function is so simple that a modified exhaustive search is used. The routine repeatedly bounds the solution using successively finer increments. The function is invoked by simply typing inverse_inv(y) where y is the number for which the angle is desired. For example

```
»inverse_inv(0.017673)

ans =  21.1267
```

The number y must be between 0 and 1, and the angle is returned in degrees.

8.B MATLAB Program for Drawing Spur Gear Profile.

8.B.1 Overview

In this appendix, the equations given in Section 8.12 are programmed in the m-file *geardr.m* to draw the hob, gear tooth, and gear. The equations for the standard hob (no protuberance) in Example 8.5 are programmed. The input variables for the routine are given in the following. Values in brackets are the default values.

```
Dp        = Diametral pitch [20] 20
ar        = Addendum constant for rack [1.0]
br        = Dedendum constant for rack [1.25]
phi       = Pressure angle in degrees [20]
rt        = Radius of tip of rack tooth [0.02]
rf        = Radius of fillet of rack tooth [0.04]
N         = Number of teeth on gear [10]
ag        = Addendum constant for gear [1.0]
ptregion1 = Number of points in region of rack tip land [30]
ptregion2 = Number of points in region of rack tip radius [30]
ptregion3 = Number of points in region of rack flank [40]
ptregion4 = Number of points in region of rack base radius [30]
ptregion5 = Number of points in region of rack bottom land [40]
```

The program outputs the geometry constants for the gear and draws half of a rack tooth, the gear tooth generated by the rack, and the gear.

8.B.2 Results from Sample Run of *geardr.m*

The sample run solves the problem in Example 8.5. The input data is given in Table 8.B.1 and the graphical results are given in Figs. 8.B.1 through 8.B.3. Notice that the resulting gear teeth are severely undercut. The numbers of points used in the different regions of the rack are somewhat arbitrary; however, if the condition indicated in Fig. 8.34 is not completely eliminated, the use of more points will usually resolve the problem.

Table 8.B.1: Input Data and Numerical Results from *geardr.m*

```
        Gear Tooth Generation Program
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (geardr.dat): manual.dat
Enter value of diametral pitch [10]: 10
Enter value of addendum constant for rack [1.25]: 1.25
Enter value of dedendum constant for rack[1.0]: 1.0
Enter value of pressure angle in degrees [20]: 20
Enter radius of tip of rack tooth [0.02]: 0.03
Enter radius of fillet of rack tooth [0.04]: 0.04
Enter number of teeth on gear [10]: 8
Enter value of addendum constant for gear [1.0]: 1
Enter number of points in rack tip land [30]: 30
Enter number of points in rack tip radius [30]: 30
Enter number of points in rack flank [40]: 40
Enter number of points in rack base radius [30]: 30
Enter number of points in rack bottom land [30]: 30

       Reinput data? [y/n]: n

                  Gear parameters
     ar       br       ag        p         r
  1.250    1.000    1.000    0.314    0.400

     ro      phi       lt        lb        rt        rf
  0.500   20.000    0.024    0.028    0.030    0.040
```

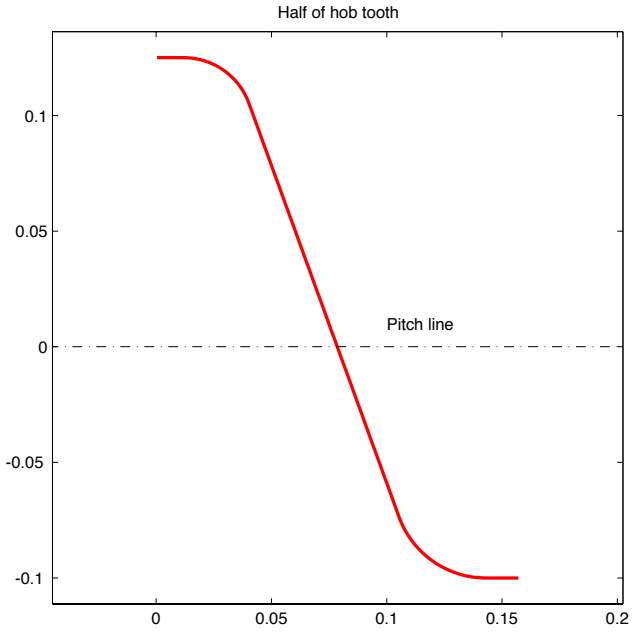


Fig. 8.B.1: Drawing of half of rack tooth.

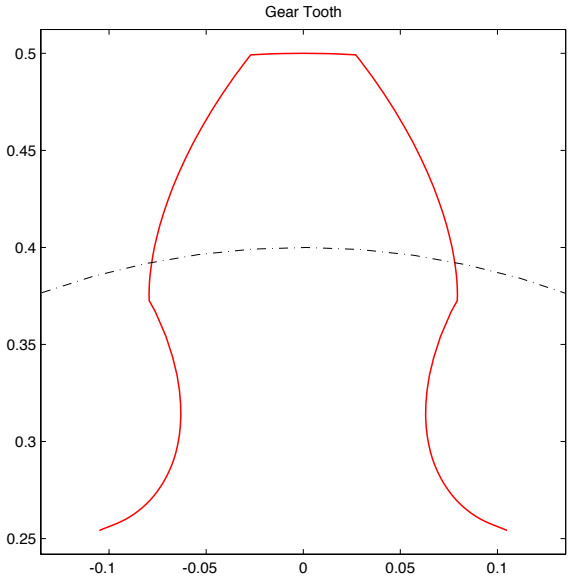


Fig. 8.B.2: Drawing of gear tooth.

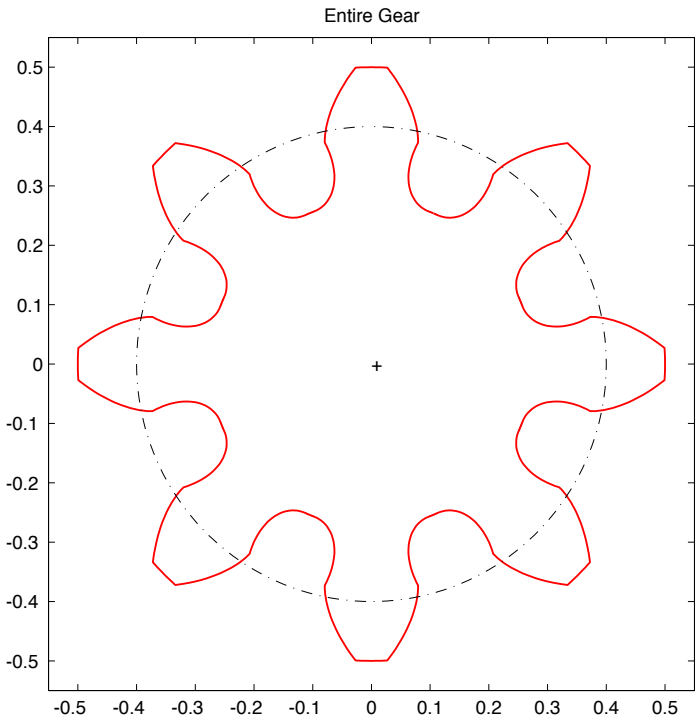


Fig. 8.B.3: Drawing of entire gear.

8.C MATLAB Program for Drawing Conjugate Tooth Form.

8.C.1 General Conjugate Tooth Forms

The fundamental law of gearing requires that when two gears are in contact, the angular velocity ratio is

inversely proportional to the lengths of the two line segments created by the intersection of the common normal to the two contacting surfaces and the line of centers. This ratio is constant if the common normal intersects the centerline at a fixed point, the pitch point. The tooth forms satisfying this condition are said to be conjugate. The flat sided rack and involute tooth form are one example of conjugate tooth forms; however, there are an infinite number of other tooth forms which can be conjugate. In this section, we will generalize the procedure given in Section 8.12 of the text book to develop a procedure for finding the tooth form which is conjugate to a general tooth form.

The information in Appendix 8.C was originally contained in the text book; however, it was removed because of page constraints. Therefore, the entire development is given here followed by the MATLAB program which will draw a conjugate gear.

8.C.1.1 Required Geometric Parameters

Several parameters must be known about both gears to determine the unknown tooth form which is conjugate to the known tooth form. These parameters include the pitch radii, tooth numbers, and gear type (i.e., internal or external), and a mathematical function for the gear tooth form on the known gear. If the function is not known directly, it is possible to fit a spline to a set of points describing the tooth form. Ultimately, it is necessary to be able to compute the normal vector to the known gear tooth at each location.

8.C.1.2 Determination of the Point of Contact

Assume that the known gear is gear 2 and the unknown gear is gear 3. Each gear will have a coordinate system attached as shown in Fig. 8.C.1, and the local gear geometry will be defined relative to the coordinate system fixed to each gear. To satisfy the fundamental law of gearing, the line normal to the tooth surfaces must pass through the pitch point as shown by line *AP* in Fig. 8.C.1. The line segment *AP* is a straight line which has the following equation:

$$y = mx + b$$

Here, m is the slope of the line which is the direction of the normal to the known gear at point *A* and *b* is the *y* intercept. An expression for *b* can be found by recognizing that the line passes through point *A*. Therefore,

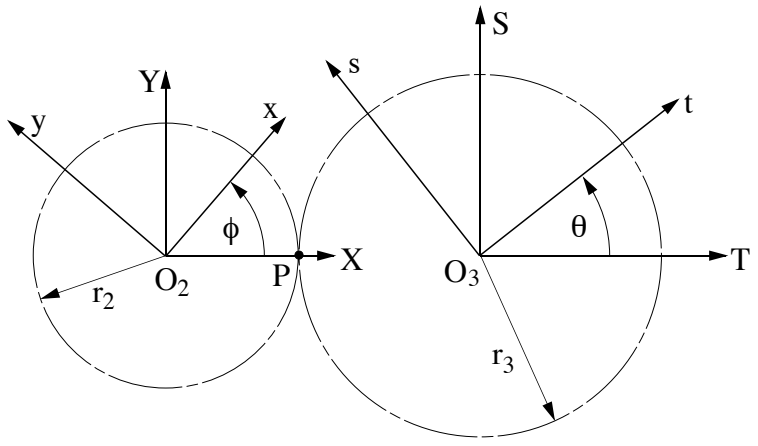


Fig. 8.C.1: Coordinate systems on two gears.

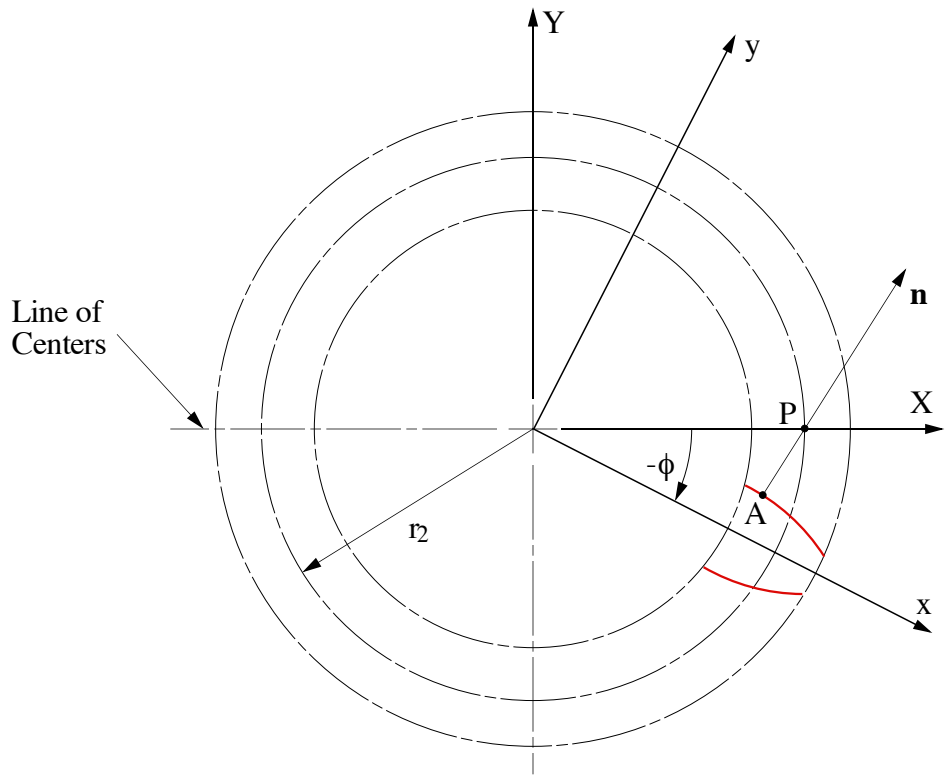


Fig. 8.C.2: Line through pitch point and normal to tooth profile.

$$y_A = mx_A + b$$

or

$$b = y_A - mx_A$$

If the slope of the normal is represented by

$$n_A = \left(\frac{n_{Ay}}{n_{Ax}}\right)$$

then an expression for the line segment is given by

$$y = (x - x_A)n_A + y_A \tag{8.C.1}$$

Two special cases exist for the line. The first occurs when the line is horizontal. Then $y = y_A$ regardless of x. When the line is vertical, then $x = x_A$ for all values of y.

Note that in Eq. (8.C.1), we assume that the components of the normal vector are known. If only an equation for the tooth profile is known, we can obtain the normal to the curve at any point by differentiation. For example, if the tooth profile is given by

$$y = F(x),$$

then the slope of the normal vector is given by

$$n = -\frac{1}{dy/dx}$$

where the derivative is evaluated at the point of interest. If x and y are given as parametric expressions, for example,

$$y = r\sin\theta$$
$$x = r\cos\theta$$

then the slope of the normal can be computed from

$$n = -\frac{1}{dy/dx} = -\frac{1}{\left(\frac{dy}{d\theta}\right)\Big/\left(\frac{dx}{d\theta}\right)}$$

Referring to Fig. 8.C.2, the line AP through P can be written relative to the coordinate system attached to gear 2 as

$$x_P = r_2\cos\phi$$
$$y_P = -r_2\sin\phi \qquad (8.C.2)$$

where r_2 is the pitch circle radius of gear 2. The negative sign on y_P is present because ϕ is negative. Substituting Eqs (8.C.2) into Eq. (8.C.1), gives

$$-r_1\sin\phi = (r_1\cos\phi - x_A)n_A + y_A$$

or

$$r_1\sin\phi + r_1 n_A\cos\phi + y_A - x_A n_A = 0 \qquad (8.C.3)$$

In a typical problem, both x_A and y_A will be specified. This will correspond to the contact point location for both gears, although the x_A and y_A specified will be relative to gear 2. We must find the coordinates relative to gear 3 to find the point on gear 3 which is conjugate to the point on the gear. However, to do this, we must first find the angle ϕ.

The angle ϕ can be found using the procedures given in Chapter 3 of the text book. To begin, make the following substitutions

$$\cos\phi = \frac{\left[1 - \tan^2\left(\frac{\phi}{2}\right)\right]}{\left[1 + \tan^2\left(\frac{\phi}{2}\right)\right]}$$

$$\sin\phi = \frac{\left[2\tan\left(\frac{\phi}{2}\right)\right]}{\left[1 + \tan^2\left(\frac{\phi}{2}\right)\right]}$$

$$\tau = \tan\left(\frac{\phi}{2}\right)$$

$$A = y_A - x_A n_A$$

$$B = r_1$$

$$C = r_1 n_A$$

Then, the equation to be solved is

$$A + B\sin\phi + C\cos\phi = 0 = A + B\left[\frac{2\tau}{1+\tau^2}\right] + C\left[\frac{1-\tau^2}{1+\tau^2}\right]$$

and the solution is

$$\tau = \frac{-B + \sqrt{B^2 - A^2 + C^2}}{A - C}$$

and

$$\phi = 2\tan^{-1}\tau$$

Note that all points on the known gear may not be possible choices for a contact point. If the candidate point chosen is an impossible contact point, $B^2 - A^2 + C^2$ will be negative.

To locate the angle ϕ for all possible points x_A and y_A, it is only necessary to begin at one end of the known contour and increment x until the other end is reached. The increments of x need not be uniform.

8.C.1.3 Coordinate Transformations

Once the point of contact is located, it becomes necessary to transform the coordinates from gear 2 to gear 3. The transformation will involve the following parameters:

C_d = center distance for two gears

θ_0 = initial angle for axis t on gear 3 when the angle ϕ is zero.

N_2 = number of teeth on gear 2

N_3 = number of teeth on gear 3

The center distance is given by

$$C_d = r_2 + ir_3 \tag{8.C.4}$$

where "i" is equal to 1 for an external gear and -1 for an internal gear.

The initial angle θ_0 for the t axis on gear 3 is π minus the angle that subtends an arc which is one half of the tooth thickness measured at the pitch circle. This angle is equal to

$$\theta_0 = \pi - \pi / N_3 = \pi\left(1 - \frac{1}{N_3}\right) \tag{8.C.5}$$

The angles θ and ϕ are related by the ratio of the pitch radii. As ϕ increases, θ decreases for external gears. The resulting relationship is

$$\theta = -i\phi\frac{r_2}{r_3} + \theta_0 \tag{8.C.6}$$

The coordinates must be transformed through four sets of coordinate systems: local xy to global XY, to global TS, and finally to local ts. Referring to Fig. 8.C.8, the xy and XY coordinate systems pertain to gear 2; whereas the TS and ts systems refer to gear 3. The x coordinate axis is along the center line of the tooth in gear 2 while the - t axis is along the centerline of the contacted gear on gear 3. The three successive transformations are given in the following.

$$\begin{Bmatrix} X \\ Y \end{Bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix}; \quad \begin{Bmatrix} T \\ S \end{Bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} X \\ Y \end{Bmatrix} - \begin{Bmatrix} C_d \\ 0 \end{Bmatrix}; \quad \begin{Bmatrix} t \\ s \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} T \\ S \end{Bmatrix}$$

The overall transformation is

$$\begin{Bmatrix} t \\ s \end{Bmatrix} = \begin{bmatrix} \cos(\theta - \phi) & \sin(\theta - \phi) \\ -\sin(\theta - \phi) & \cos(\theta - \phi) \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} + \begin{Bmatrix} -C_d \cos\theta \\ C_d \sin\theta \end{Bmatrix} \tag{8.C.7}$$

These equations define the conjugate tooth form relative to gear 3.

Example 8.C.1 (Conjugate Tooth Form for Straight Toothed-Gearing)

Problem:

One tooth form which has been used on very large gears such as the ring gear on draglines is a straight toothed form. This is the same form as is used on a simple rack except that the pitch curve is a circle instead of a straight line. Therefore, the conjugate tooth form is not an involute. For the problem, assume that gear 2 has a pitch diameter (d_2) of 20 feet and the diametral pitch (D_p) is 5 teeth per foot of pitch diameter. The tooth surface is inclined with the centerline at an angle of $\phi = 25^\circ$. The mating gear (gear 3) is an external gear with 30 teeth (N_2). The addendum constant (a_2, a_3) for each gear is 1, and the dedendum constant (b_2, b_3) is 1.2. Find the tooth form which is conjugate to gear 2 so that there will be a constant velocity ratio between the two gears.

Solution:

To find the conjugate tooth form, we must first find an expression for the coordinates of points on the gear tooth and for the components of the normal vectors. Figure 8.C.3 shows an enlarged view of gear 2. The equations for the gear are similar to those for the hob in Example 8.5.

Before developing the equations, it is useful to compute several parameters. These are:

A_2 = Addendum of gear 2 = $\dfrac{a_2}{D_p} = \dfrac{1}{5} = 0.2\ ft$

B_2 = Dedendum of gear 2 = $\dfrac{b_2}{D_p} = \dfrac{1.2}{5} = 0.24\ ft$

γ = tooth angle = 25°

ℓ_t = gear 2 tooth thickness at tip = $\ell_t = \dfrac{\pi}{2D_p} - 2A_2 \tan\gamma = \dfrac{\pi}{2(5)} - 2(0.2)\tan 25 = 0.128\ ft$

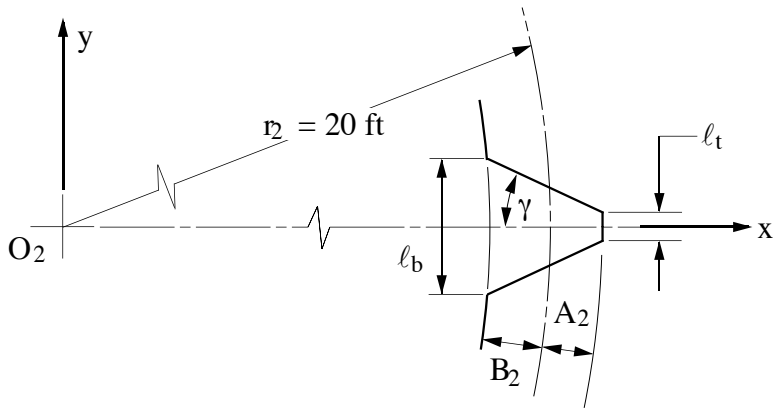


Fig. 8.C.3: One tooth from pin gear.

ℓ_b = gear 2 tooth thickness at tip = $\ell_b = \frac{\pi}{2D_p} + 2B_2\tan\gamma = \frac{\pi}{2(5)} + 2(0.24)\tan 25 = 0.538\ ft$

From that figure,

$$\begin{Bmatrix} x \\ y \end{Bmatrix} = \begin{Bmatrix} r_2 - B_2 + \beta \\ \frac{\ell_b}{2} - \beta\tan\gamma \end{Bmatrix}; \quad \begin{Bmatrix} n_x \\ n_y \end{Bmatrix} = \begin{Bmatrix} \sin\gamma \\ \cos\gamma \end{Bmatrix}; \quad 0 \le \beta \le (A_2 + B_2) \tag{8.C.8}$$

We need consider only one side of the driving tooth because only one side will contact the corresponding tooth on gear 3 for a given direction of rotation. We can reflect the tooth about its centerline to find the other half.

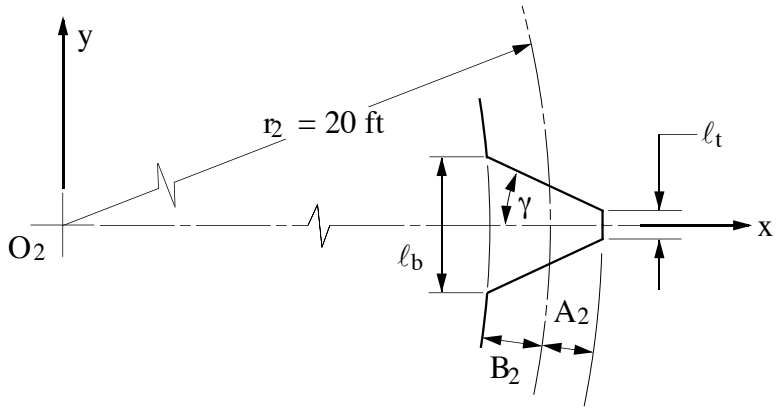


Fig. 8.C.3: One tooth from pin gear.

The number of teeth on gear 2 is

$$N_2 = d_2 D_p = 20(5) = 100$$

The pitch radius of gear 3 is given by

$$r_3 = \frac{N_3}{N_2} r_2 = \frac{30}{100} 10 = 3\ ft$$

and the center distance is given by Eq. (8.C.4) as

$$C_d = r_2 + ir_3 = 10 + (+1)(3) = 13\ ft$$

The initial angle θ_0 is given by Eq. (8.C.5) as

$$\theta_0 = \pi\left(1 - \frac{1}{N_2}\right) = \pi\left(1 - \frac{1}{30}\right) = 3.037\,rad$$

To find the conjugate gear form, it is only necessary to increment β from 0 to $(A_2 + B_2)$ and compute the (x, y) coordinates of the points and normals using Eq. (8.C.8). The angle ϕ corresponding to the selected point can then be found by solving Eq. (8.C.3) using the procedure given above.

The angles θ for a given value of ϕ is given by Eq. (8.C.6):

$$\theta = -i\phi\frac{r_1}{r_2} + \theta_0 = -\frac{2}{3}\phi + 0.209$$

Knowing θ and ϕ, the coordinates of the conjugate point on gear 3 are given by Eq. (6.39) or

$$\begin{Bmatrix} t \\ s \end{Bmatrix} = \begin{bmatrix} \cos(\theta - \phi) & \sin(\theta - \phi) \\ -\sin(\theta - \phi) & \cos(\theta - \phi) \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix} + \begin{Bmatrix} -C_d \cos\theta \\ C_d \sin\theta \end{Bmatrix}$$

Once the values of *t, s* on gear 3 are known for each value of *x, y*, on gear 2, the tooth form on gear 3 can be computed. Clearly, this procedure is best done using a computer program to determine the tooth profile of gear 3.

A MATLAB program called *arb2th.m* is written to satisfy the conditions of the problem, but no provision is made to eliminate undercutting if a small number of teeth is used on the generated gear. The program uses the same terms defined for standard gears. The definition of the input variables for the routine are given in the following. Values in brackets are the default values.

```
Dp         = Diametral pitch [5]
N1         = Number of teeth on gear 2 [100]
N2         = Number of teeth on gear 3 [30]
a2         = Addendum constant for gear 3 [1.0]
npoints    = Number of points on gear 2 [200]
a2         = Addendum constant for gear 2 [1.0]
b2         = Dedendum constant for gear 2 [1.25]
gamma      = Flank angle for gear 2 in degrees [25]
```

The program outputs the geometry constants for the gear and draws the generating tooth, the conjugate tooth, the generating gear, the conjugate gear, and the two gears in mesh.

The sample run solves the problem in Example 8.C.1. The input data is given in Table 8.C.1 and the graphical results are given in Figs. 8.C.4 through 8.C.8. When the program is run, by pressing return, the gears can be shown in several positions of mesh.

Table 8.C.1: Input Data and Numerical Results from *arb2th.m*

```
Conjugate Gear Tooth Generation Program
```

```
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (arb2th.dat): manual.dat
Enter value of diametral pitch [5]: 5
Enter number of teeth on gear 1 [100]: 100
Enter number of teeth on gear 2 [30]: 30
Enter addendum constant for gear 2 [1.0]: 1
Enter number of points in gear 1 tooth [200]: 200
Enter value of addendum constant for gear2 [1.0]: 1
Enter value of dedendum constant for gear2[1.2]: 1.2
Enter value of flank angle for gear 1, degrees [25]: 25
Enter factor for external or internal gear (1 ext; -1 int) [1]: 1

                  Gear parameters
     r2       r3       a3        N2         N3
  10.000    3.000    1.000   100.000    30.000

     Cd       t0       A3
  13.000    3.037    0.200
      »
```

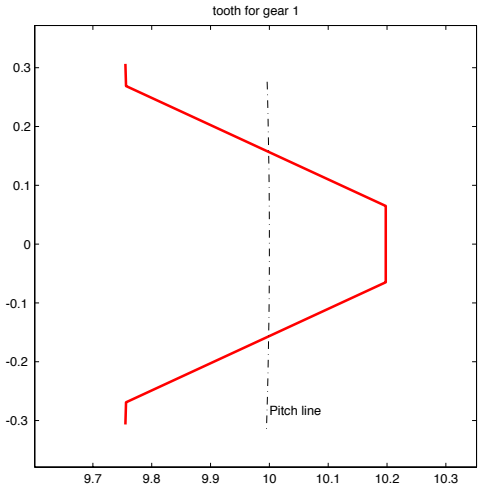


Fig. 8.C.4: Generating tooth form.

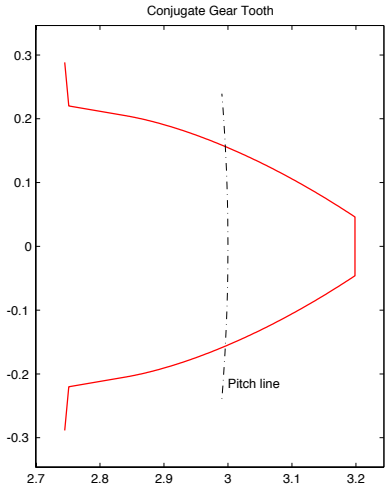


Fig. 8.C.5: Conjugate tooth form.

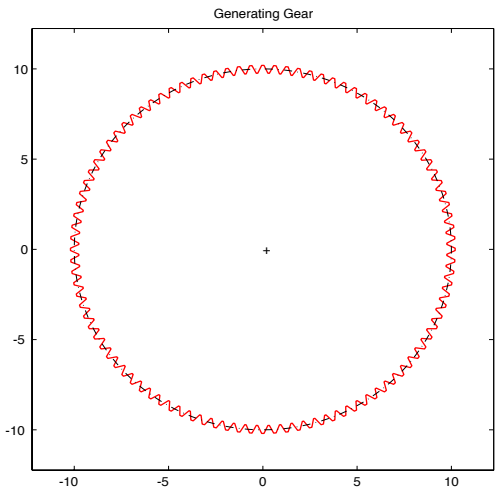


Fig. 8.C.6: Drawing of generating gear.

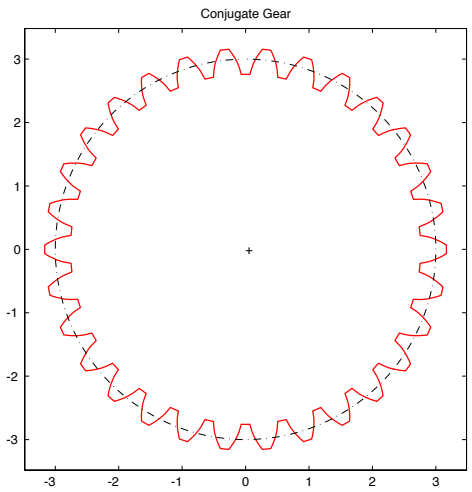


Fig. 8.C.7: Drawing of conjugate gear.

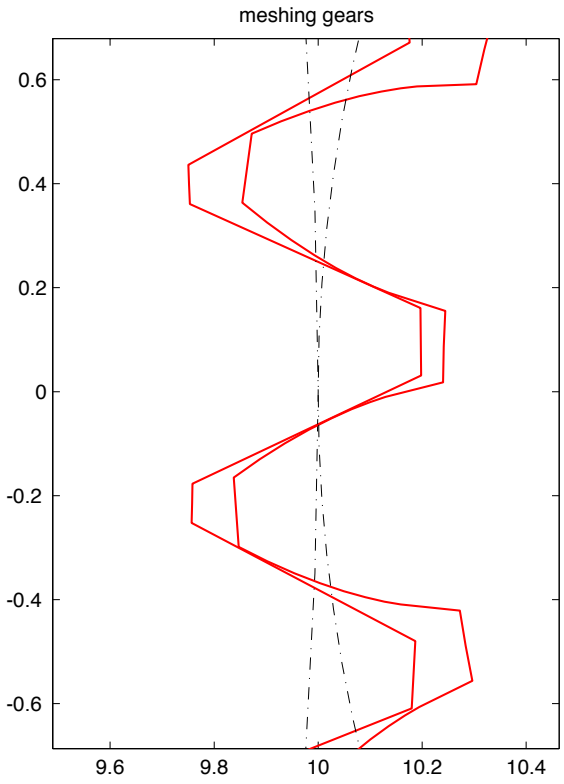


Fig. 8.C.8: Gears in mesh.

8.D MATLAB Program for Drawing Rack Envelope (*rackmotion.m*)

8.D.1 Overview

The program, rackmotion.m draws the envelope of a rack or hob as it cuts a gear blank. This program was used to draw Fig. 8.22. The input variables for the routine are given in the following. Values in brackets are the default values.

```
Dp        = Diametral pitch [10] 20
ar        = Addendum constant for rack [1.25]
br        = Dedendum constant for rack [1.1]
phi       = Pressure angle in degrees [20]
rt        = Radius of tip of rack tooth [0.02]
rf        = Radius of fillet of rack tooth [0.04]
N         = Number of teeth on gear [10]
ag        = Addendum constant for gear [1.0]
npos      = Number of rack positions [15]
```

The program plots the geometry of the rack, moves it relative to the gear and replots the position. This is repeated for *npos* positions.

8.D.2 Results from Sample Run of *rackmotion.m*

The input data is given in Table 8.D.1 and the graphical results are given in Figs. 8.D.1 and 8.D.2.

Table 8.D.1: Input Data and Numerical Results from *rackmotion.m*

```
          Rack Motion Program

Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (rackmotion.dat): temp.dat
Enter value of diametral pitch [10]: 15
Enter value of addendum constant for rack [1.25]: 1.25
Enter value of dedendum constant for rack [1.1]: 1.1
Enter value of pressure angle in degrees [20]: 25
Enter radius of tip of rack tooth [0.02]: 0.01
Enter radius of fillet of rack tooth [0.04]: 0.02
Enter number of teeth on gear [10]: 8
Enter value of addendum constant for gear [1.0]: 1.0
Enter number of rack positions [15]: 20
```

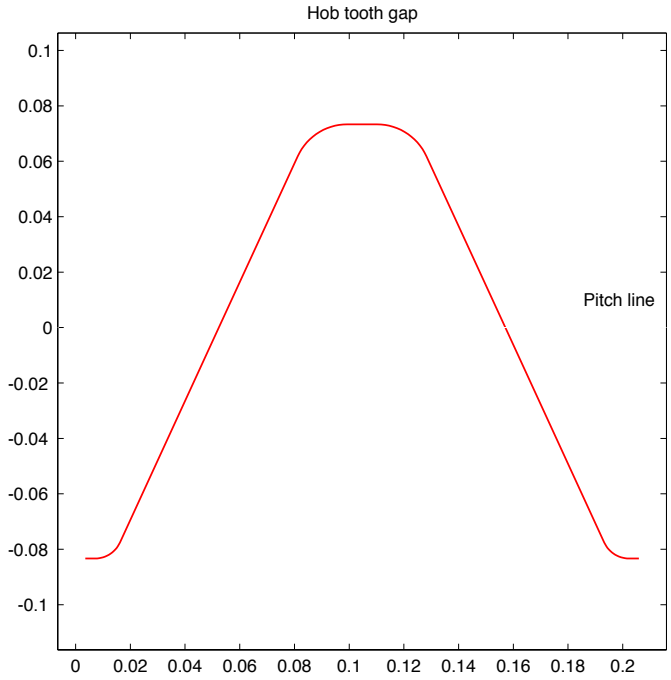


Fig. 8.D.1: Drawing of half of rack tooth.

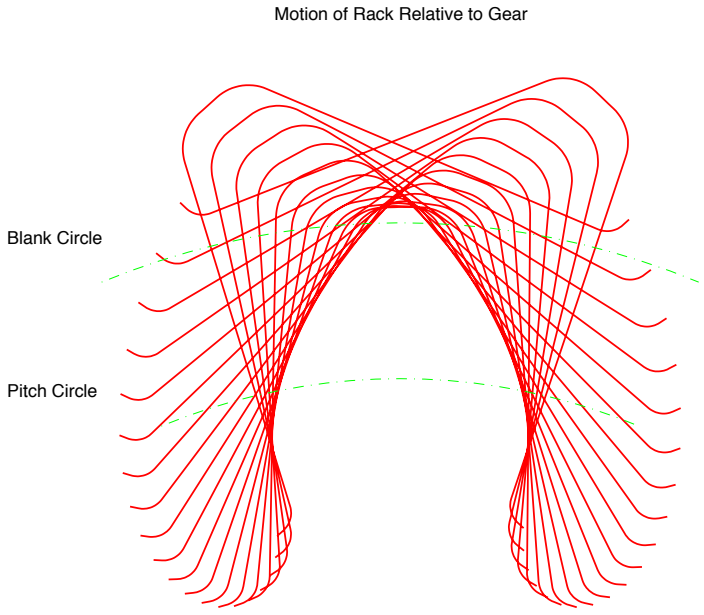


Fig. 8.D.2: Envelope developed by rack.

9.0 Programs for Chapter 9

(No programs have been written for Chapter 9)

10.0 Programs for Chapter 10

(No programs have been written for Chapter 10)

11.0 Programs for Chapter 11

(No programs have been written for Chapter 11)

12.0 Programs for Chapter 12

(No programs have been written for Chapter 12)

13.0 Programs for Chapter 13: Slider-Crank Balancing

13.A MATLAB Program for Balancing Slider-Crank Mechanism

This appendix contains a description of the MATLAB function routine for balancing a slider-crank mechanism. The slider offset is assumed to be zero, and the crank is assumed to be able to rotate for 360˚. The source code for the functions is included on the disk included with this book.

The MATLAB function files are based on the crank-driven slider-crank routine given in Appendix 3.C. The balancing routine is called *shake.m*. The program first analyzes the slider crank mechanism for one-degree increments of the crank for position, velocity, and acceleration. The crank angular velocity is assumed to be constant. After the acceleration of the crank pin and piston are known for the different positions, the shaking force is computed at each angle increment for both the given value of the counter-balance weight and for zero counter balance. The maximum shaking force and position of the crank are determined for each counter-balance weight. Next, the optimum value of the counterbalance weight is determined using the Golden Section optimization procedure[1].

The program animates the slider-crank mechanism, and displays plots of the shaking force for the three values of the counter-balance weight. The routine shake.m calls the following other function routines: *axisadjust.m, rectangle.m*, *bushing.m*, *circle.m*, *sldcrkc.m*, *sc_angle_limits_cr.m*, and *frameline.m*. All of these routines were described in the appendices of Chapter 3. No units are assumed for the forces and lengths used in the program; however, a consistent set of units must be used with the input variables. The input data generally follow the nomenclature given in Fig. 13.A.1.

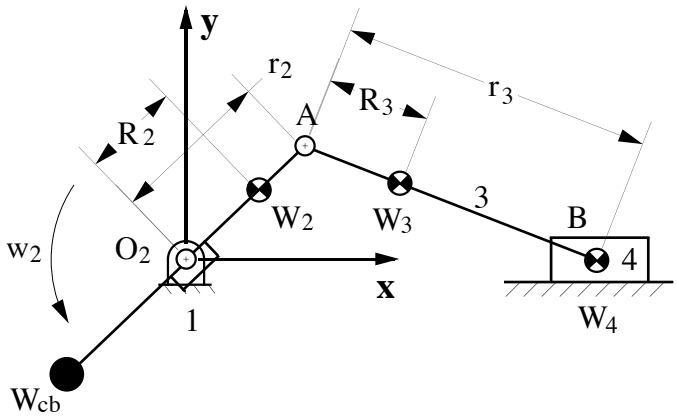


Fig. 13.A.1: Nomenclature used in shaking force analysis program.

The input variables are:

```
shake.dat = file to which input data are to be written (and from which input data
            can be read).  Any name can be used for this file.
tt        = number of cycles for animation
r2        = length of crank
r3      = length of coupler
mode    = assembly mode [-1 or +1]
```

---

[1] Arora, Jasbir S. Introduction to Optimum Design, McGraw-Hill Book Company, New York, 1989, pp. 297-300.

```
w2      = crank angular velocity (rad/sec)
W2      = weight of crank
R2      = distance from crank rotation axis to center of mass
W3      = weight of coupler
R3      = distance from crank pin to center of mass of coupler
W4      = weight of piston
Wcb     = weight of counter balance (assumed to be on opposite side of
          crank axis from A, and at a distance of A from the crank axis)
g       = acceleration of gravity in consistent units.
```

The results from the sample analysis used to solve Example 13.5 are given in Table 13.A.1. The graphical results are given in Figs. 13.A.2 and 13.A.3.

Table 13.A.1: Results for Examples 13.5

```
    Slider-Crank Analysis Program to Determine Shaking Forces
Enter 1 for file input and 2 for interactive input [1]: 2
Enter input file name (shake.dat): manualdat
Enter number of cycles (rev) [3]: 3
Enter the crank length [4]: 4
Enter the coupler length [14]: 14
Enter the assembly mode (+1 or -1) [1]: 1
Enter the angular velocity of crank(rad/sec) [4]: 104.72
Enter the weight of the crank [1.875]: 28.04
Enter distance from O2 to crank CG [2]: 4
Enter the weight of the coupler [34]: 9.71
Enter distance from coupler CG to point A [4]: 14
Enter the weight of the piston [20]: 20
Enter the counter balance weight [47.85]: 47.85
Enter the acceleration of gravity [386]: 386

    tt      r2      r3      mode    w2
  3.000   4.000  14.000   1.000  104.720

    W2      R2      W3      R3      W4
 28.040   4.000   9.710  14.000  20.000

    Wcb     g
 47.850  386.000

The maximum shaking force corresponding to Wcb=0 is    7527.36
at theta =       0.00


The maximum shaking force corresponding to Wcb is    2495.42
at theta =      98.18

The optimum counter balance weight is      45.87
The corresponding maximum shaking force is    2314.66
at theta =     258.18

Repeat animation? y/n [y]: n
Repeat animation? y/n [y]: n
```

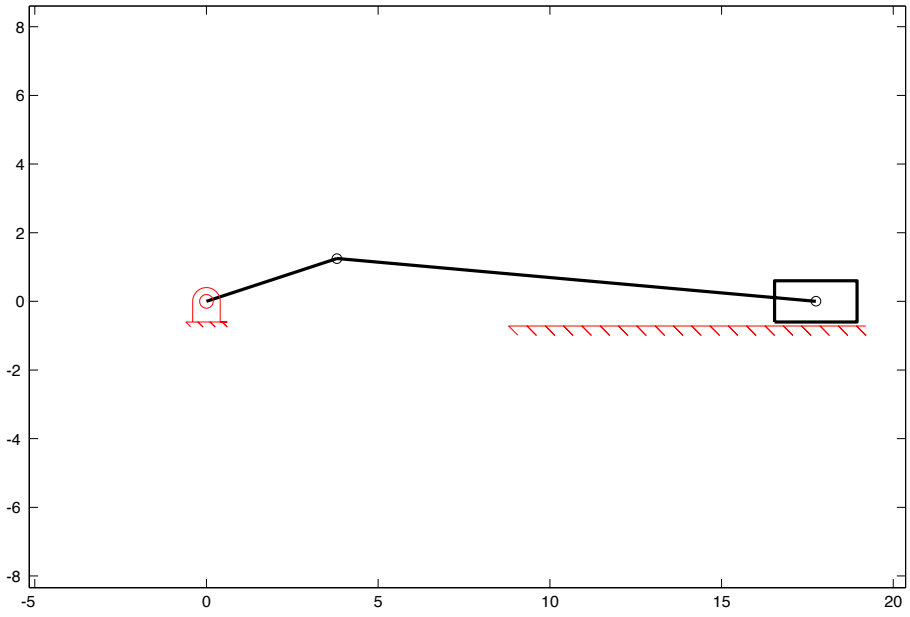


Fig. 13.A.1: Slider-crank mechanism analyzed

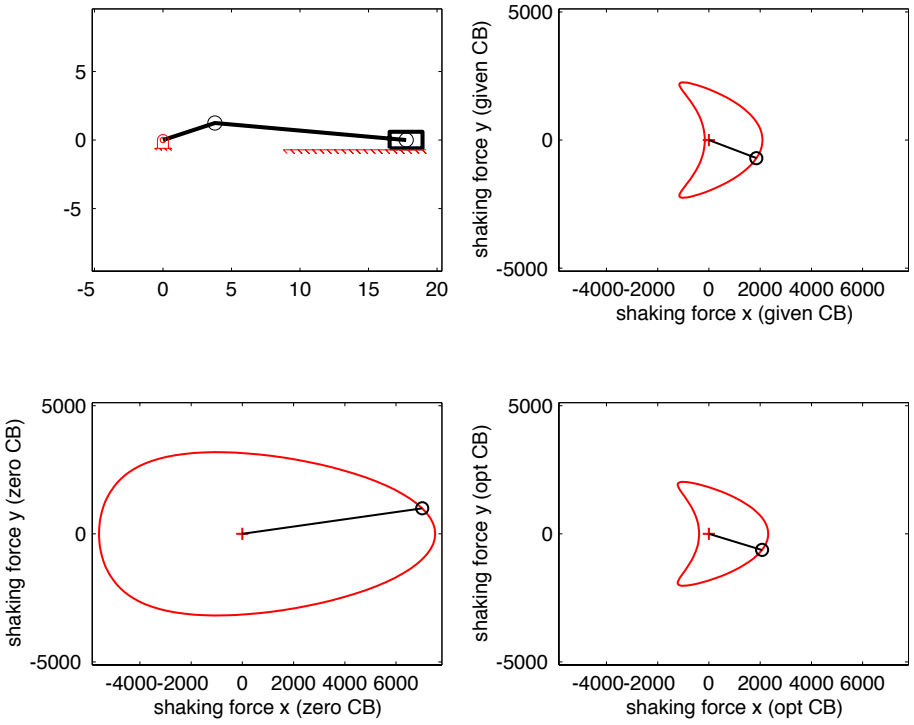


Fig. 13.A.2: Shaking force diagrams for zero, given, and optimum counter-balance weights

14.0 Brief Overview of MATLAB for Applications in Kinematics

14.1 Introduction

MATLAB is a high performance interactive software package used for scientific and engineering numeric computation. The name MATLAB stands for *matrix laboratory*. MATLAB integrates numerical analysis, matrix computation, signal processing, and graphics in an easy-to-use environment. Complex numerical problems can be solved using MATLAB without actually writing a program. The objective of this handout is to help the reader understand the main concepts of MATLAB and begin to use MATLAB in different applications. However, the reader is encouraged to use the on-line help facility in order to find more detailed information concerning the use of MATLAB. The capabilities of MATLAB go far beyond those presented in this brief overview, and consequently the reader is encouraged to consult the *MATLAB User's Guide*. The *Student Edition of MATLAB* can be used for solving all of the problems in the book entitled *Kinematics, Dynamics, and Design of Machinery* by K. Waldron and G. Kinzel.. The programs with that book are based on Version 5.0 of MATLAB, and this overview is also based on that version.

14.2 M-files

MATLAB can execute a sequence of statements stored in disk files. Such files are called M-files because they have the extension ".m" attached to the filename. For example, *test.m* indicates that the file name is *test* and the type file is *M-file*. Much of the work with MATLAB will be in creating and refining M-Files. The M-file is an ASCII file and can be edited in different ways on SGI workstations, DOS computers (PCs) and Macintosh computers.

There are two types of M-files: *script files* and *function files*. A script file consists of a sequence of normal MATLAB statement. If the file has the filename, say, *rotate.m*, then the MATLAB command *rotate* will cause the statements in the file to b executed. Variables in a script file are global and will change the value of the environment.

Script files are often used to enter data into a large matrix, in such a file, entry errors can be easily edited out. If, for example, one enters in a diskfile *data.m*

```
A = [
1 2 3 4
5 6 7 8
];
```

then the MATLAB statement data will cause the assignment given in data.m to be carried out.

An M-file can reference other M-files, including referenceing itself recursively.

Function files provide extensively to MATLAB. You can create new functions specific to your problem which will have the same status as other MATLAB functions. Variables in a function file are local.

We first illustrate with a simple example of a function file:

```
function P = prodsqr (A, B)
% PRODSCR Product of the square of two matrices
P = A^2*B^2
```

Memory will be used more efficiently if A is overwritten with the result:

```
function A = prodsqr (A, B)
```

```
% PRODSCR Product of the square of two matrices
A = A^2*B^2
```

This should be placed in a diskfile with filename prodsqr.m (corresponding to the function name). The first line declares the function name, input arguments, and output arguments; without this line the file would be a script file. Then a MATLAB statement.

z = prodsqr(x, y), for example, will cause the variables x and y to be passed to the variables A and B in the function file with the output result being passed out to the variable z. Since variables in a function file are local, their names are independent of those in the environment.

A function may also have multiple output arguments. For example:

```
function [mean, stdev] = stat(x)
% STAT Mean and standard deviation
%       For a vector x, stat(x) returns the mean and standard deviation of x.  For a
%       matrix x, stat(x) returns two row vectors containing, respectively, the mean
%       and standard deviation of each  column.
[m, n] = size(x);
if m == 1
    m = n; % handle case of a row vector
end
mean = sum(x)/m;
stdev = sqrt (sum(x.^2)/m - mean.^2)
```

Once this is placed in a diskfile *stat.m*, a MATLAB command *[xm, xd] = stat (x)*, for example, will assign the mean and standard deviation of the entries in the vector x to xm and xd, respectively. Single assignments can also be made with a function having multiple output arguments. For example, *xm = stat (x)* will assign the mean of *x* to *xm*.

This function illustrates some of the MATLAB features that can be used to produce efficient code. Note, for example, that x:^2 is the matrix of squares of the entries of x, that sum is a vector sum, that sqrt is a scalar function, and that the division sum(x)/m is a matrix-scalar operation.

The % symbol indicates that the rest of the line is a comment; MATLAB will ignore the rest of the line. However, the first few comment lines, which document the M-file, are available to the on-line help faculity and will be desplayed if, for example, *help stat* is entered.

14.3 Manipulating Data with MATLAB

MATLAB works with essentially only one kind of object - a rectangular numerical matrix with possibly complex entries; all variables represent matrices. In some situations, special meaning is attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or one column, which are vector.

Matrices can be introduced into MATLAB in several different ways:

- Entered by an explicit list of elements,
- Generated by built-in statements and functions,
- Created in M-files (see below),
- Loaded from extended data files.

Data are manipulated in MATLAB in matrix and vector form. A matrix may be entered by typing at the MATLAB prompt >>

```
>>A = [1 2 3; 4 5 6; 7 8 9];
```

The ";" sign placed after the statement suppresses the display of the data in matrix A on the screen. Generally speaking, it is better to use the ";" sign after statements unless you have a reason to see the output resulting from a specific statement. This matrix can be displayed by typing A. MATLAB will display:

```
1     2                    3
3     4                    5
6     7                    8
```

A column vector may be entered by typing:

```
>>V = [1; 2; 3];
```

and can be displayed by typing V. MATLAB will respond with:

```
1
2
3
```

A row vector may be entered by typing:

```
>>V = [1 2 3 4];
```

and can be displayed by typing V. MATLAB will respond with:

```
1 2 3 4
```

A row vector can be changed to a column vector by transposition. For example, the vector V presented above is a row vector. *V'* vector is a column vector. Practice the following example:

```
>>V=[1 2 3 4];  % row vector
>>U=V';         % column vector
>>U
```

Here, the symbol "%" stands for comment. The statements following it on the same line are not executed.

To generate a row vector of time intervals 0.01 seconds starting from 0 to 10 seconds, you should type:

```
>>[t] = 0:0.01:10;
```

You can use vectors as arguments of functions. For example, for the time intervals input above you can calculate and plot a mathematical function defined as presented below:

```
>>[x] = exp(0.5*t).*sin(20*pi*t);
```

The period after exp(0.5*t) means an element by element multiplication. You can plot now [x] as a function of [t] by typing:

```
>>plot(t,x,'r')
```

MATLAB will respond with the plot of [x] as a function of time [t] shown in Fig. 1. Note in Fig. 1 that the plot axes are unlabled and the graph is untitled. Procedures for placing lables and titles will be discussed later in this manual. As you can see MATLAB is a very strong tool that allows you to do a very large variety of applications. Using MATLAB, you can visualize very easily any function previously defined and

any set of data.

When listing a number in exponential form (e.g. 2.34e-9), blank spaces must be avoided. Listing entries of a large matrix is best done in an M-file, where errors can be easily edited away.

The built-in functions *rand, magic*, and *hilb*, for example, provide an easy way to create matrices with which to experiment. The command *rand(n)*, and *rand(m,n)*, will create an n-by-n and m-by-n matrix, respectively, with randomly generated entries; *magic(n)* will create an integral n-by-n matrix which is a magic square (rows and columns have common sum); *hilb(n)* will create the n-by-n Hilbert matrix, a very ill-conditioned matrix. Matrices can also be generated with a for-loop.

Individual matrix and vector entries can be referenced with indices inside parentheses in the usual manner. For example, A(2,3) denotes the entry in the second row and third column of a matrix A and x(3) denotes the third coordinate of a vector x.

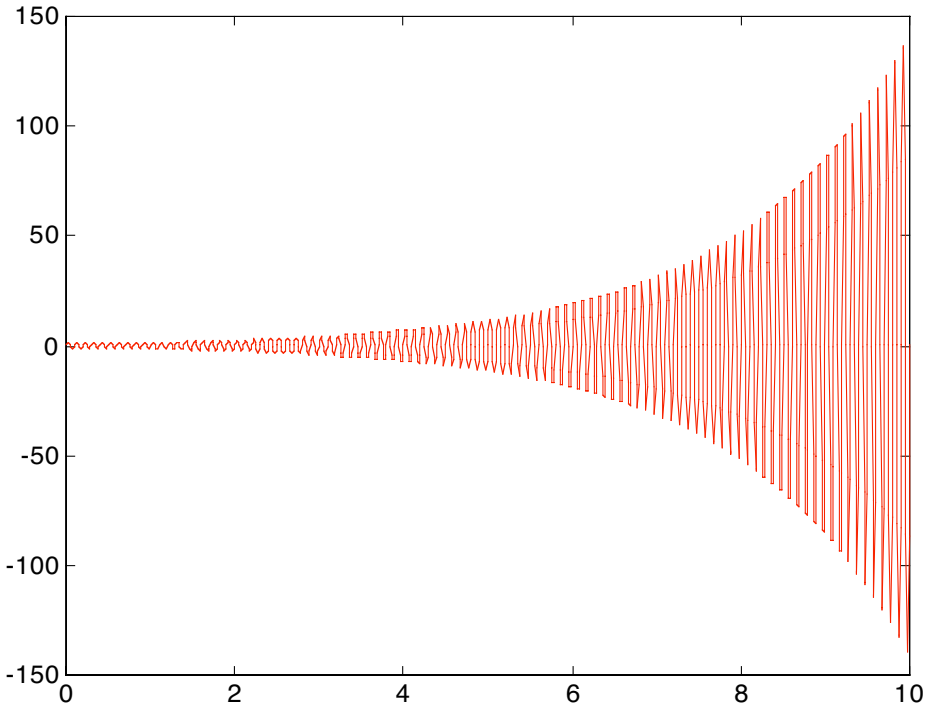


Fig. 14.1: Plot of `exp(0.5*t).*sin(20*pi*t)` versus t

14.3.1 Pointwise operations with matrices

Element operations between two matrices of the same size are possible with MATLAB. The element-wise operations are presented below:

$$C = A.*B \Rightarrow c_{ij} = a_{ij} * b_{ij}$$
$$C = A.\backslash B \Rightarrow c_{ij} = b_{ij} / a_{ij}$$
$$C = A./B \Rightarrow c_{ij} = a_{ij} / b_{ij}$$
$$C = A.\hat{}\ B \Rightarrow c_{ij} = a_{ij}^{b_{ij}}$$

$C = A.\text{\^{}}\, e \quad \Rightarrow c_{ij} = a_{ij}^{e}$

Again notice that the period following the first matrix signifies an element by element operation.

14.3.2 Matrixwise operations with matrices

Similar operations are also available on a matrix basis. In particular, the following apply on a *matrix* basis if a period is not used:

+	addition
-	subtraction
*	multiplication
^	power
'	transpose
\	left division
/	right division

These matrix operations apply also to scalars (1-by-1 matrices). If the sizes of the matrices are incompatible for the matrix operation, an error message will result, except in the case of scalar-matrix operations in which case each entry of the matrix is operated on by the scalar.

The "matrix division" operations deserve special comment. If *A* is an invertible square matrix and *b* is a compatible column or row vector, then

x = A\b is the solution of *A*x=b*

and

x=b/A is the solution of *x*A=b*.

In the left division, if A is square, then it is factored using Gaussian elimination and these factors used to solve *A*x=b*. If A is not square, it is factored using Houserholder orthogonalization with column pivoting and the factrors used to solve the under- or over-determined system in the least squares sense. Right division is defined in terms of left division by *b/A = (A'\b')'*.

14.4 Statements, expressions in MATLAB

14.4.1 General statements

MATLAB is an expression language, i.e., the expressions you type are interpreted and evaluated. In their basic form, the MATLAB statements operate like those in most computer languages. Typically, expressions are of the form:

variable = expression

or simply

expression.

Expressions are usually composed from operators, functions, and variable names. Evaluation of the expression produces a matrix, which is then displayed on the screen and assigned to the variable for future use. If the variable name and = sign are omitted, a variable *ans* (for answer) is automatically created to which the result is assigned.

A statement is normally terminated with the carriage return. However, a statement can be continued to the next line with three or more periods followed by a carriage return. On the other hand, several statements can be placed on a single line if separated by commas or semicolons.

MATLAB is case-sensitive in the names of commands, functions, and variables. For example, *solveX* is not the same as *solvex*.

The command *who* will list the variables currently in the workspace. A variable can be cleared from the workspace with the command *clear variablename*. The command *clear* alone will clear all nonpermanent variables.

When one logs out or exists MATLAB, all variables are lost. However, invoking the command *save* before exiting causes all variables to be written to a diskfile named *matlab.mat*. When one later reenters MATLAB, the command *load* will restore the workspace to its former state.

A runaway display or computation can be stopped on most machines without leaving MATLAB with CTRL-C.

The permanent variable *eps* (epsilon) gives the machine precision - about 10^{-16} on most machines. It is useful in determining tolerances for convergence of iterative processes.

14.4.2 Matrix building functions

Convenient matrix building functions are:

eye	identity matrix
zeros	matrix of zeros
ones	matrix of ones
diag	diagonal matrix
triu	upper triangular part of a matrix
tril	lower triangular part of a matrix
rand	randomly generated matrix
hilb	Hilbert matrix
magic	magic square
toeplitz	see help toeplitz in MATLAB

For example *zeros(m,n)* produces an m-by-n matrix of zeros and *zerox(n)* produces an n-by-n one; if *A* is a matrix, then *zeros(A)* produces a matrix of zeros of the same size as *A*.

If *x* is a vector, *diag(x)* is the diagonal matrix with *x* down the diagonal; if *A* is a square matrix, then *diag(A)* is a vector consisting of the diagonal of *A*.

Matrices can be built from blocks. For example, if *A* is a 3-by-3 matrix, then

B = [A, zeros(3,2); zeros(2,3), eye(2)]

will build a certain 5-by-5 matrix.

14.4.3 Relational operators

The legal relational operators in MATLAB are

==	equals
~=	not equals
<	less than
<=	less than or equal to
>	greater than

>= greater or equal to

Note that "=" is used in an assignment statement while "==" is used in a relation. Relations may be connected or quantified by the logical operators

& and
| or
~ not.

When applied to scalars, a relation is actually the scalar 1 or 0 depending on whether the relation is true or false. When applied to matrices of the same size, a relation is a matrix of 0's and 1's giving the value of the relation between corresponding entries.

Scalar examples are:
3<5 => ans = 1
3>5 => ans = 0
3==5 => ans = 0
3==3 => ans = 1
A matrix example is:
For

```
»a=rand(5)
a =
    0.2190    0.3835    0.5297    0.4175    0.5269
    0.0470    0.5194    0.6711    0.6868    0.0920
    0.6789    0.8310    0.0077    0.5890    0.6539
    0.6793    0.0346    0.3834    0.9304    0.4160
    0.9347    0.0535    0.0668    0.8462    0.7012
```

and

```
»b=triu(a)
b =
    0.2190    0.3835    0.5297    0.4175    0.5269
         0    0.5194    0.6711    0.6868    0.0920
         0         0    0.0077    0.5890    0.6539
         0         0         0    0.9304    0.4160
         0         0         0         0    0.7012
```

Then,

```
»a==b
```

give

```
ans =
     1     1     1     1     1
     0     1     1     1     1
     0     0     1     1     1
     0     0     0     1     1
     0     0     0     0     1
```

14.4.4 If-statements

The simplest *if* statement has the form

```
if{relation}
   {statements};
end
```

As an example of *if* statement

```
if a <= 0
   b = a;
```

```
end
```

A relation between matrices is interpreted by *while* and *if* to be true if each entry of the relation matrix is nonzero. Hence, if you wish to execute *statement* when matrices *A* and *B* are equal, you would type

```
if A == B
    statement
end
```

but if you wish to execute statement when *A* and *B* are not equal, you would type

```
if A == B else
    statement
end
```

Note that that the statements

```
if A ~= B
    statement
end
```

will not execute *statement* unless each element of *A* differs from the corresponding element of *B*.

General *If-then-else* constructs are also possible in MATLAB. The general form of the *if-then-else* construct is

```
if{relation 1}
   {statements 1};
elseif{relation 2}
   {statements 2};
   .
   .
   .
elseif{relation N}
   {statements N};
else
   {statements};
end
```

14.4.5 For-loops

In MATLAB *for*-loops have the form

```
for {var} = {row vector or counter values}
   {statements};
end
```

For example the statement

```
for i = 1:n, x(i) = i^2, end
```

or

```
for i = 1:n
x(i) = i^2;
end
```

will produce a certain n-vector, and the statement

```
for i = n:-1:1, x(i) = i^2, end
```

will produce the same vector but the elements will be computed in reverse order. As you can see based on these statements, MATLAB statements operate like those in most computer languages. Remenber that MATLAB is case-sensitive in the names of commands, functions, variables.

14.4.6 While-loops

While-loops are possible in MATLAB, and they have the general form

```
while {relation}
   {statements};
end
```

As an example, let us consider the calculation of the sum of the first N natural numbers.

```
N = 100;
n = 0;
sum = 0;
while n < N,
   n = n+1;
   sum = sum+n;
end
```

You can realize from this example that *N* as variable is not the same as *n*. MATLAB is a *case sensitive* computer package. Therefore, be very carefull with the cases of the variables.

14.5 Functions in MATLAB

14.5.1 Scalar functions

Certain MATLAB functions operate essentially on scalars, but operate element-wise when applied to a matrix. The most common such function are:

sin	asin	exp	abs	round
cos	acos	log(natural log)	sqrt	floor
tan	atan	rem(remainder)	sign	ceil

14.5.2 Vector functions

Other MATLAB functions operate essentially on a vector (row or column), but act on an m-by-n matrix (m≥ 2) in a column-by-column fashion to produce a row vector containing the results of each column. A few of these functions are:

max	sum	median	any
min	prod	mean	all
sqrt	norm	std	

For example, the maximum entry in a matrix *A* is given by *max(max(A))* rather than *max(A)*. The magnitude of the vector x is given by *norm (x)*

14.5.3 Matrix functions

Much of MATLAB's power comes from its matrix functions. Some of the useful ones are:

eig	eigenvalues and eigenvectors
chol	cholesky factorization

svd	singular value decomposition
inv	inverse
lu	LU factorization
qr	QR factorization
hess	hessenberg form
schur	schur decomposition
rref	reduced row echelon form
expm	matrix exponential
sqrtm	matrix square root
poly	characteristic polynomial
det	determinant
size	size
norm	1-norm, 2-norm, F-norm, ∞ norm
cond	condition number in the 2-norm
rank	rank

MATLAB functions may have single or multiple output arguments. For example,

y = eig(A), or simply eig(A)

produce a column vector containing the eigenvector of A while

[U,D] = eig(A)

produces a matrix *U* whose columns are the eigenvectors of *A* and a diagonal matrix *D* with the eigenvalues of A on its diagonal.

14.5.4 Submatrices and colon notation

Vectors and submatrices are used often in MATLAB to achieve fairly complex data manipulation effects. "Colon notation", which is used both to generate vectors and reference submatrices, and subscripting by vectors are keys to efficient manipulation of these objects. Creative use of these features permits one to minimize the use of loops (which slows down MATLAB) and to make code simple and readable. Special effort should be made to become familiar with them.

The expression 1:5 (met earlier in *for* statements) is actually the row vector [1 2 3 4 5]. The numbers need not be integers nor the increment one. For example,

0.2:0.2:1.2
gives [0.2 0.4 0.6 0.8 1.0 1.2], and

5:-1:1 gives [5 4 3 2 1].

The following statements will, for example, generate a table of sines.

```
x=[0.0: 0.1: 2.0]';
y = sin(x);
[x, y]
```

Note that since sin operates entry-wise, it produces a vector y from the vector x. The colon notation can be used to access submatrices of a matrix. For example,

A(1:4,3) is the column vector consisting of the first four entries of the third column of A.

A colon by itself denotes an entire row or column:

A(:,3) is the third column of A, and A(1:4,:) is the first four rows.

Arbitrary integral vectors can be used as subscripts:

A(:,[2 4]) contains as columns, columns 2 and 4 of A.

Such subscripting can be used on both sides of an assignment statement:

A(:, [2 4 5]) = B(:, 1:3) replaces columns 2, 4, 5 of A with the first three columns of B.
Note that the entire altered matrix A is printed and assigned.

Columns 2 and 4 of A can be multiplied on the right by the 2-by-2 matrix [1 2;3 4] by:

```
A(:,[2, 4]) = A(:, [2,4])*[1 2;3 4]
```

Once again, the entire altered matrix is printed and assigned.

If x is an n-vector, what is the effect of the statement x = x(n:-1:1)?

14.5.5 User defined functions

It is possible to define functions in MATLAB. This feature of MATLAB greatly enhances the power of the system. A function is an M-file whose *name* is the same as the name of the function. Let us consider the following example of a MATLAB function:

```
function y = sinus(t)
% calculation of a sine wave of amplitude 20
y = 20*sin(0.1*pi*t);
```

As you can see, the first line of the M-file function contains the definition of the function. The general form of the definition is:

```
function {variable} = {function name}( {arguments} )
```

Again, comments lines begin with the sign %. All functions should begin with comment lines describing how to use the function.. It is not necessary to use an entire line to type comments. You can use the following pattern:

```
y = 20*sin(0.1*pi*t); % scale y
```

The variables used by a function that do not appear in the function statement are local unless they have been declared global. Let us consider again the previous example in which we will introduce a local variable *a* and an external variable *b* which is passed from the MATLAB environment together with the vector *t*.

```
function y = sinus(t,b)
a = 20.0;
y = a*sin(b*pi*t);
```

In the example presented above, the *name* of the MATLAB function is *sinus*. The name of the M-file must be also *sinus*. If you want to use this function to calculate *y* for the values of *t* contained into the vector *t,* use the following statements:

```
[t] = 0:0.01:10;
```

```
b = 0.1;
y = sinus(t,b);
plot(t,y);
```

Running this example you will obtain a sine plot of amplitude 20 for an angle range of 0 - 180 degrees; however, the horizontal axis will range from 0 to 10 corresponding to the range of y.

14.6 MATLAB utility commands

14.6.1 help

help, by itself, lists all primary help topics. Each primary topic corresponds to a directory name on MATLABPATH. "*help topic*" gives help on the specified topic. The topic can be a command name or a directory name. If it is a command name, *help* displays information on that command. For example, to the command:

```
>>help sin
```

MATLAB will respond with:

```
SIN(X) is the sine of the elements of X.
```

If the topic is a directory name, *help* displays the Table-Of-Contents for the specified directory.

14.6.2 plot, subplot, semilogx, semilogy, loglog, grid, title, xlabel, ylabel

plot(X,Y) plots the vector X versus the vector Y. If X or Y is a matrix, then the vector is plotted versus the rows or columns of the matrix, whichever line up. plot(y) plots the columns of Y versus their index. If Y is complex, plot(y) is equivalent to *plot(real(Y),imag(Y))*. In all other uses of *plot*, the imaginary part is ignored.

plot(x1,y1,x2,y2) is another way of producing multiple lines on the plot. Using the previous command, the two lines will be of the same type. If one wants to draw different types of lines, the following command: *plot(x1,y1,'*',x2,y2,'+')* will produce a plot using * for the first curve and + for the second curve. Various line types, plot symbols and colors may be obtained with *plot(x,y,s)* where S is a 1, 2, or 3 character string made from the following characters:

```
y    yellow      .    point
m    magenta     o    circle
c    cyan        x    x-mark
r    red         +    plus
g    green       -    solid
b    blue        *    star
w    white       :    dotted
k    black       -.   dashdot
      --         dashed
```

As an example of using *plot* command we can write an M-file called plot1.m. You may type the following statements in this M-file:

```
[x]=0:0.1:10;
[y1]=1-exp(-0.5*x);
[y2]=exp(-0.5*x);
plot(x,y1,'g*',x,y2,'r+')
```

```
title('Y1 and Y2 versus X')
xlabel('X')
ylabel('Y1 (*) and Y2 (+)')
grid on
```

In the section *M-files*, you have already seen how you can create an M-file. You can find a definition of the commands *title, xlabel, ylabel* and *grid* at the end of this section. To run the M-file in the matlab window at the >> prompt just type *plot1* and hit the *return* key. MATLAB will respond with the plot in Fig. 2.

subplot(m,n,p), or *subplot(mnp)*, breaks the figure window into an m-by-n matrix of small plots, selects the p-th plot for the current plot, and returns the plot handle. The plots are counted along the top row of the figure window, then along the second row, etc. For example,

```
subplot(1,2,1), plot(fig1)
subplot(1,2,2), plot(fig2)
```

plots Fig. 1 on the left half of the window and Fig. 2 on the right half. As an example of using subplot, we can write an M-file named subplot1.m with the following content:

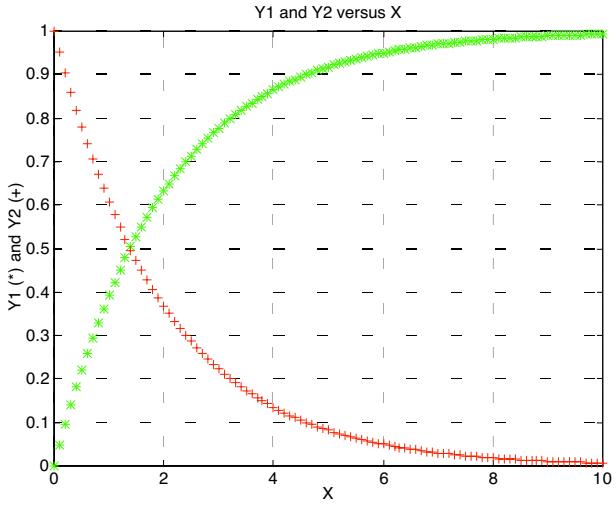


Fig. 14.2: Example of using *plot* command

```
[x1]=0:.1:10;
[y1]=1-exp(-0.5*x1);
[y2]=exp(-0.5*x1);
subplot(1,2,1),plot(x1,y1,'b')
grid on
xlabel('x1')
ylabel('y1')
title('y1 versus x1')
subplot(1,2,2),plot(x1,y2,'r')
grid on
xlabel('x1')
ylabel('y22')
title('y2 vs x1')
```

When you run this M-file (by typing *subplot1* and pressing return), you will obtain a plot of y1 vs x on the left half of the window and a plot of y2 vs x on the right half of the window as in the plot presented in Fig. 3.

The following gives the definitions for some of the useful plotting support functions.

semilogx(Y,X) plots vector Y over vector X, using the semilog scale on the X-axis.
semilogy(Y,X) plots vector y over vector y, using the semilog scale on the Y-axis.
loglog(Y,X) plots vector y over vector x, using the log scale on both X and Y axes.
grid on adds grid lines to the current axes. *grid off* takes them off.
title('Title') adds *Title* at the top of the current axis.
xlabel('Xlabel') adds *Xlabel* bellow the X-axis on the current axis.
label('Ylabel') adds *Ylabel* bellow the Y-axis on the current axis.

See also *polar, clf, clc, axis, axes, hold, mesh,* and *contour* using the *help* command.

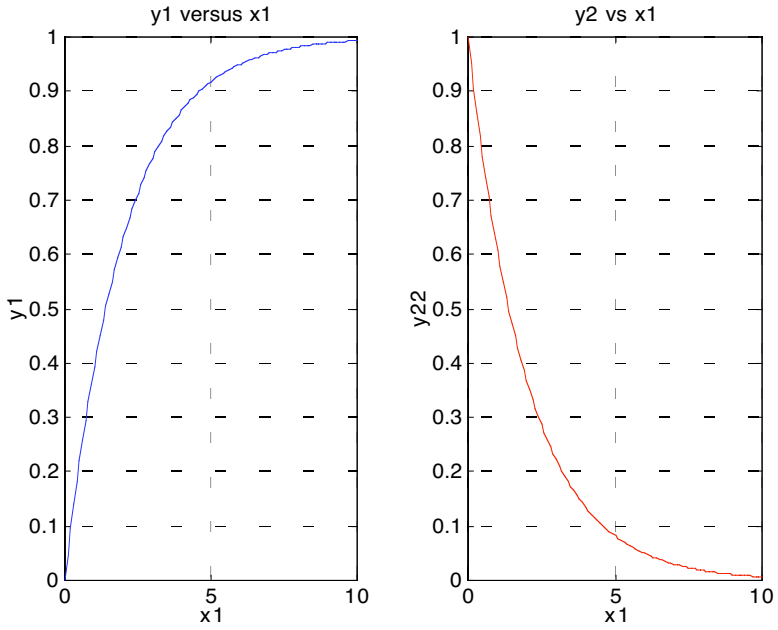


Fig. 14.3: Example of using *subplot* command

14.6.3 Text strings, error messages, input

Text strings are entered into MATLAB surrounded by single quotes. For example,

s = 'this is a test'

assigns the given test string to the variable *s*.

Text strings can be displayed with the function disp. For example:

disp ('this message is hereby displayed')

Error messages are best displayed with the function error:

error ('Sorry, the matrix must be symmetric')

since it causes execution to exit the M-file.

In an M-file the user can be prompted to interactively enter input data with the function input. When, for example, the statement

iter = input ('Enter the number of iterations: ')

is encountered, the prompt message is displayed and execution pauses while the user keys in the input data. Upon pressing the return key, the data is assigned to the variable *iter* and execution resumes.

14.7 Animation Graphics

14.7.1 Introduction

MATLAB uses an object-oriented system of graphics commands called Handle Graphics. Handle Graphics defines a set of graphics objects, such as lines, surfaces, and text, and provides mechanisms to manipulate the characteristics of these objects to develop the desired results. This permits the graphics to be developed from objects rather than relying on the high level plotting routines.

The graphics system is based on a parent-child relationship where the children take on the attributes of the parents. Therefore, once the attributes of the parent routine are defined, these attributes need not be defined for the children unless the attributes are different for the children. For example, if the line color is defined for the parent, the same line color will be used for the children.

Using Handle Graphics, a realistic animation can be achieved if the number of vectors drawn is modest, (perhaps 10) and no shaded images are involved. If a large number of vectors are used or shaded panels are required, a mechanism will not be displayed properly.

Animation in MATLAB can be achieved using the routine AXES and some of its children. AXES are the parents of large number of routines; however, we will used only two (text, line). In the following, we will discuss those commands in MATLAB which will permit a mechanism to be drawn and animated. This will use a very small subset of the graphics capabilities used in MATLAB

To draw a figure on the screen, it is necessary to identify the area (viewport) on the screen where the figure is to appear, properly label the figure and axes, scale the axes, and draw the figure within the viewport. Each of these operations and the corresponding commands are described in the following.

14.7.2 Identifying the Viewport and Plotting Axes

The function *axes* create axes in arbitrary positions and identifies the location on the screen where the drawing is to be done. For example axes('position', [x0, y0, width, height]) opens up a drawing area at the specified location and returns a handle to it. The location on the screen is specified by the left, bottom, width, and height of the viewport. It specifies the location and size of the side of the axis box, relative to the lower-left corner of the figure window, in normalized units where (0,0) is the lower-left corner and (1.0,1.0) is the upper-right. The function *axes* also defines a default set of axes for the drawing.

The command *h1=axes* creates a set of default axes (whole screen) with handle H current. After *axes* is invoked, all drawing objects will be associated with *h1*. To define a number of different graphics objects on the same screen, *axes* should be invoked with different handles and the objects defined accordingly. The actual object properties and legal property values can be identified by executing *get(h1)* and *set(h1)*, respectively. For animation in kinematics, the most commonly used properties are

```
position          ycolor
box               ylabel
linewidth         fontname
title             fontangle
```

```
xcolor            fontsize
xlabel
```

The properties can be defined in the *axes* arguments directly or on separate lines. If the properties are included in the argument list, the property name is included in single quotes followed by the value(s) for the property. For example,

```
h1=axes('position', [0.6 .55 .4, .4],'box', 'on','xcolor', 'k','ycolor', 'k');
```

defines *axes* with the handle *h1*. The viewport for the axes is located in the upper left hand quadrant of the screen, a box is drawn around the axes, and the color of both the x and y axes is black. The property 'box' can be either 'on' or 'off' depending on whether the box is to be drawn or not.

Linewidth gives the width of a line drawn. The width of a line which has linewidth = 100 is approximately 35 mm wide.

Xcolor and ycolor give the colors for the x and y axis, respectively. The color can be specified by mixing the colors red, green, and blue using a three component vector [rgb] where the components range between 0 and 1. If component 1 is 0, no green is present. If component 1 is 1, maximum green is present. For the common colors, symbols can be used by writing the letter in single quotes as was done above. The symbols and corresponding colors are shown in Table 4.

It is most convenient to label the axes and give a title to the graph in separate commands. We can then use separate fonts, colors, and styles if we want. For example,

```
xlabel ('theta', 'color', 'r', 'fontname', 'times', 'fontangle', 'italic', 'fontsize',24)
```

labels the x axis with *theta*, using a 24 point times font in italics in the color red.

In general we will want a white background for all plots. This facilitates printing the results. This is done by the command

```
set (gcf, 'color', 'w').
```

after the *axes* command is executed. Here gcf is the handle for the current graphics figure. To set the background color to black we would use

```
set (gcf, 'color', 'k')
```

Table 14.4. Color representations in MATLAB

Symbol	Color (RGB)
c	cyan [0 1 1]
m	magenta [1 0 1]
y	yellow [1 1 0]
r	red [1 0 0]
g	green [0 1 0]
b	blue [0 0 1]
w	white [0 0 0]
k	black [1 1 1]

14.7.3 Scaling the Axes

Once the axes are drawn, if no further scaling is done, the maximum range for the axes will be [0 1]. Scaling and appearance is controlled using the routine *axis*. This routine can be used several different ways. Basic scaling is done using

axis ([xmin, xmax, ymin, ymax])

where xmin and ymin are the minimum values for the x and y axes, respectively, and xmin and ymin are the maximum values.

Axis equal

sets the aspect ratio so that equal tick mark increments on the x-,y- and z-axis are equal in size. This makes a circle look like a circle, instead of an ellipse.

Axis square

makes the current axis box square.

Axis off

turns off all axis labeling, tick marks, and background

Axis on

turns on the axis labeling, tick marks, and background.

14.7.4 Line Objects

A drawing will be made up of line objects which are defined by the *line* command. *Line (x,y)* adds the line in vectors x and y to the current axes. Lines are children of *axes* and can have a number of properties defined after x and y are specified. To identify all of the line object properties and their current values, execute *get (h)* where h is the line handle. To identify the legal properties for line, execute *set (h)*. The properties which we will use to draw mechanisms are:

Color
Erasemode
Linestyle
Linewidth
Marker:
Markersize
Xdata
Ydata

The properties color and linewidth are the same as defined for *axes*.
Erasemode identifies the mode with which the line is erased, and this controls dynamic redrawing. Three erase modes are available.

• none- MATLAB does not erase previous instances of the object when it is moved.

• Background - MATLAB erases the previous instance of the object by redrawing it in the background color. This mode erases the object and everything below it (such as other lines or gridlines)

• xor - MATLAB erases only the object and is usually used for animation.

Linestyle give the line style for the line drawn. MATLAB has a limited number of line styles which can be automatically generated, and these are shown in Table 5.

Table 14.5. Automatic Linestyles in MATLAB

Symbol	Line Style
_	solid line (default)
--	dashed line
:	dotted line
-.	dashed dotted line
none	no line

The property marker identifies the type of marker which is to be placed at each data point. The markers which are available are shown in Table 6. When drawing mechanisms, typically we will use 'none' for drawing links; however, it is convenient to use the marker 'o' to represent revolute joints. MATLAB can draw a marker faster than it can draw a circle from an array of points by executing line. The property *markersize* gives the marker size in points.

The line drawn when *line* is executed is really a polyline. This means that a series of line segments will be drawn, and the end points of the line segments are identified by the arrays *xdata* and *ydata*. The properties xdata and ydata give the x and y arrays, respectively, of the line segments which are to be drawn. The dimension of xdata can be any number larger than 0, but both arrays must have the same number of elements. If xdata and ydata have no elements, no line is drawn. If they have only one element each, nothing will appear on the screen unless a marker is specified. If a marker is specified, the center of the marker will appear at the coordinates specified.

Table 14.6. Types of Markers in MATLAB

Marker Specified	Description
+	plus sign
o	circle
*	asterisk
.	point
x	cross
square	square
diamond	diamond
^	upward pointing arrow
>	right pointing arrow
<	left pointing arrow
v	downward pointing arrow
pentagram	five-pointed star
hexagram	six-pointed star
none	no marker (default)

When the line object is to be animated, it is common to identify as many of the properties of the line object as possible outside the animation loop in the matlab program. For example,

```
joint=line('xdata', [], 'ydata', [], 'marker', 'o','markersize', 10, 'erasemode', 'xor', 'color', 'r')
```

defines the line object with the handle joint, and sets of arrays with no elements in xdata and ydata. The circle marker will be used with a size of 10 points. The erase mode is xor meaning that the previous instance of joint will be erased in the xor mode before the current instance is drawn. The line will be drawn in red. As another example,

```
coupler=line('xdata', [], 'ydata' ,[], 'linewidth' ,3, 'erasemode', 'xor','color', 'g');
```

defines a line object with handle coupler and no elements in xdata and ydata. No marker will be used. The erase mode is xor meaning that the previous instance of coupler will be erased in the xor mode before the current instance is drawn. The line will be drawn in green.

14.7.5 Updating Line Objects

To give the appearance of animation, we must be able to draw a line on one position, erase the line and draw it in another position. Erasemode indicated how (or if) the previous line is to be erased. Generally, we will use *xor* or *none* depending on whether we want the line to be erased or not.

Each new instance of the line object is invoked by the *set* command. The command *set* sets object properties. Set(h,'PropertyName1',PropertyValue1,'PropertyName2',PropertyValue2,...) sets multiple property values with a single statement for the object with the handle h. Note that it is permissible to use property/value string pairs, structures, and property/value cell array pairs in the same call to *set*.

Generally, in kinematics, the properties which we will want to change are *xdata* and *ydata*. For example, the line object *coupler* identified above could be updated using

```
set(coupler,'xdata',[Bx(i) Cx(i)],'ydata',[By(i) Cy(i)]);
```

where [Bx(i) Cx(i)] and [By(i) Cy(i)] are the current x and y values, respectively, of the two points defining the line object. Each time this line is executed, the previous instance of coupler is erased, and the current instance is drawn.

14.7.6 Input functions

To introduce data into an m-file, several procedures can be used. One is to use assignment statements, for example

```
a = 4;
```

Another is to read variables from a file and a third is to read the information from the screen interactively. This can be done fairly simply using the structure shown in the following

```
variable=input('Enter input value [1]:  ');

if isempty(variable);

        variable=1;

end
```

When this sequence is executed, the statement 'Enter input value [1]:' is printed on the screen, and the program pauses until the user keys in a value for variable. The default value is shown in brackets. If the user simply presses return, the value for variable is returned as empty, and the default value (1 in this example) is assigned to variable.

Another means of inputting information is through the mouse. This is especially useful for graphics oriented programs such as those involved in kinematics. To input a variable value via the mouse, execute the command *ginput* for graphical input through the mouse. The syntax for the command is

```
[x, y] = ginput (n).
```

This command gets N points from the current axes and returns the x- and y-coordinates in length N vectors X and Y. The cursor can be positioned using a mouse (or by using the Arrow Keys on some systems). Data points are entered by pressing a mouse button or any key on the keyboard except carriage return, which terminates the input before N points are entered.

The command

```
[x, y] = ginput
```

gathers an unlimited number of points until the return key is pressed.

Typical logic for using the *ginput* command would be

```
disp('   ')
disp('  Use mouse to locate B2')
[x, y] =ginput (1);
```

When this set of statements is executed, a line is skipped on the MATLAB Command Window, and the statement ' Use mouse to locate B2' is displayed. One point will be input through the mouse and stored in [x, y]. Execution of the m-file automatically continues after the mouse button is pressed.

14.7.7 Miscellaneous Drawing Commands

Two commands are discussed elsewhere in this manual, but they will be identified here again because they are important for the appearance of the figures. The first is

```
drawnow;
```

This routine flushes the graphics buffer to ensure that all objects have been drawn. This is important because the system does not automatically draw objects until the drawing buffer if full. Thus, without executing this command, only partial figures might be drawn.

The second routine is used to clear the graphics figure before the graphics part of the program is executed. This is done with the routine

```
clf;
```

In addition, we should also execute

```
clear all;
```

which clears all arrays.

A sample graphics routine which animates the sine function shown in Fig. 17 is shown below. A more complex example which uses many of the routines discussed here is described in the appendix to this manual.

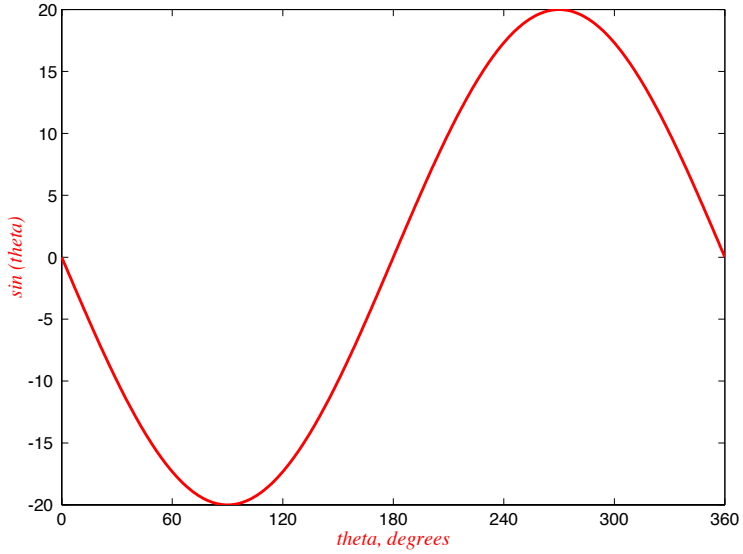


Fig. 14.4: Sine wave to be animated

```
% Sample program which animates a sine wave

clf;
clear all;
axes('position', [0.1, 0.1, 0.8, 0.8],'fontsize',12, 'box', 'on', 'xcolor', 'k', 'ycolor', 'k');
xlabel ('theta, degrees', 'color', 'r', 'fontname', 'times', 'fontangle', 'italic', 'fontsize',14);
ylabel ('sin (theta)', 'color', 'r', 'fontname', 'times', 'fontangle', 'italic', 'fontsize',14);
set (gca, 'xtick', [0, 60, 120, 180, 240, 300, 360]);
sinf=line('xdata', [], 'ydata',[], 'color', 'r', 'linewidth', 3, 'erasemode', 'xor');
fact = 180/pi;
k=0;
for j = 1:1:361
       k=k+1;
       xt(k)=j-1;
       x=xt(k)/fact;
       y(k) = sin (x);
end
axis([0, 360,-20, 20]);

% Animate the results for three full cycles

sign=-1;
for ncy=1:1:6
       sign=-sign;
       for a=-sign*20:sign*.2:sign*20
              yt=a*y;
              set (sinf, 'xdata', xt, 'ydata', yt);
              drawnow
       end
end
       set (sinf, 'xdata', xt, 'ydata', yt, 'erasemode', 'none', 'linewidth', 2);
```

14.8 Summary of MATLAB Functions

A summary of the MATLAB functions are given in the following. The information can be obtained directly from MATLAB using the Help option.

14.8.1 Color (Color Control and Lighting Models)

Color controls.

```
colormap- Color look-up table.
caxis   - Pseudocolor axis scaling.
shading - Color shading mode.
```

Color maps.

```
hsv     - Hue-saturation-value color map.
gray    - Linear gray-scale color map.
hot     - Black-red-yellow-white color
          map.
cool    - Shades of cyan and magenta color
          map.
bone    - Gray-scale with a tinge of blue
          color map.
copper  - Linear copper-tone color map.
pink    - Pastel shades of pink color map.
prism   - Prism color map.
jet     - A variant of HSV.
flag    - Alternating red, white, blue,
          and black color map.
```

Color map related functions.

```
colorbar- Display color bar (color scale).
hsv2rgb - Hue-saturation-value to red-
          green-blue conversion.
rgb2hsv - Red-green-blue to hue-saturation
          -value conversion.
contrast- Gray scale color map to enhance
          image contrast.
brighten- Brighten or darken color map.
spinmap - Spin color map.
rgbplot - Plot color map.
```

Lighting models.

```
surfl   - 3-D shaded surface with
          lighting.
specular- Specular reflectance.
diffuse - Diffuse reflectance.
surfnorm- Surface normals.
```

14.8.2 Datafun (Data Analysis and Fourier Transformations)

Basic operations.

```
max       - Largest component.
min       - Smallest component.
mean      - Average or mean value.
median    - Median value.
std       - Standard deviation.
sort      - Sort in ascending order.
sum       - Sum of elements.
prod      - Product of elements.
cumsum    - Cumulative sum of elements.
cumprod   - Cumulative product of elements.
trapz     - Numerical integration using
            trapezoidal method.
```

Finite differences.

```
diff      - Difference function and
            approximate derivative.
gradient - Approximate gradient.
del2      - Five-point discrete Laplacian.
```

Vector operations.

```
cross     - Vector cross product.
dot       - Vector dot product.
```

Correlation.

```
corrcoef - Correlation coefficients.
cov      - Covariance matrix.
subspace - Angle between subspaces.
```

Filtering and convolution.

```
filter   - One-dimensional digital filter.
filter2  - Two-dimensional digital filter.
conv     - Convolution and polynomial
           multiplication.
conv2    - Two-dimensional convolution.
deconv   - Deconvolution and polynomial
           division.
```

Fourier transforms.

```
fft      - Discrete Fourier transform.
fft2     - Two-dimensional discrete
           Fourier transform.
ifft     - Inverse discrete Fourier
           transform.
ifft2    - Two-dimensional inverse
           discrete Fourier transform.
abs      - Magnitude.
angle    - Phase angle.
unwrap   - Remove phase angle jumps across
           360 degree boundaries.
fftshift - Move zeroth lag to center of
           spectrum.
cplxpair - Sort numbers into complex
           conjugate pairs.
nextpow2 - Next higher power of 2.
```

14.8.3 Demos (Demonstration and Samples)

MATLAB/Introduction.

```
expo, demo- Start up The MATLAB Expo and
            display splash screen.
expomap   - Open the MATLAB Expo Main Map
            (avoids Expo splash screen).
```

MATLAB/Matrices.

```
intro     - Introduction to MATLAB.
inverter - Demonstrate the inversion of a
           matrix.
buckydem - Connectivity graph of the
           Buckminster Fuller geodesic
           dome.
sparsity - Demonstrate effect of sparsity
           orderings.
matmanip - Introduction to matrix
           manipulation.
delsqdemo- Finite difference Laplacian on
           various domains.
sepdemo  - Separators for a finite element
           mesh.
airfoil  - Display sparse matrix from NASA
           airfoil.
```

MATLAB/Numerics.

```
funfuns  - Demonstrate functions that
           operate on other functions.
fitdemo  - Nonlinear curve fit with
           simplex algorithm.
sunspots - The answer is 11.08, what is the
question?
e2pi     - Which is greater, e^pi or pi^e?
bench    - MATLAB Benchmark.
odedemo  - Ordinary differential
           equations.
quaddemo - Adaptive quadrature.
zerodemo - Zerofinding with fzero.
fplotdemo- Plot a function.
eigmovie - Symmetric eigenvalue movie.
rrefmovie- Computation of Reduced Row
```

```
            Echelon Form.
 fftdemo  - Use of the fast finite Fourier
            transform.
 quake    - Loma Prieta Earthquake.
 census   - Try to predict the US
            population in the year 2000.
 spline2d - Demonstrate GINPUT and SPLINE
            in two dimensions.
```

MATLAB/Visualization.

```
 graf2d   - Demonstrate XY plots in MATLAB.
 graf2d2  - Demonstrate XYZ plots in
            MATLAB.
 grafcplx - Demonstrate complex function
            plots in MATLAB.
 lorenz   - Plot the orbit around the
            Lorenz chaotic attractor.
 xpsound  - Demonstrate MATLAB V4's sound
            capability.
 vibes    - Vibrating L-shaped membrane.
 xpklein  - Klein bottle demo.
 xfourier - Graphics demo of Fourier series
            expansion.
 cplxdemo - Maps of functions of a complex
            variable.
 peaks    - A sample function of two
            variables.
 membrane - Generate MathWorks's logo.
 penny    - Several views of the penny
            data.
 earthmap - View Earth's topography.
 sqdemo   - Superquadrics using UIControls.
 imagedemo- Demonstrate MATLAB V4's image
            capability.
 colormenu- Select color map.
```

MATLAB/Language.

```
 xplang   - Introduction to the MATLAB
            language.
 graf3d   - Demonstrate Handle Graphics for
            surface plots.
 hndlgraf - Demonstrate Handle Graphics for
            line plots.
 hndlaxis - Demonstrate Handle Graphics for
            axes.
```

SIMULINK/Simple Systems.

```
 simintro - A quick introduction to
            SIMULINK.
 libintro - A quick introduction to the
            SIMULINK Libraries.
 simppend - SIMULINK system modeling a
            simple pendulum.
 onecart  - SIMULINK system modeling a
            mass-spring system.
 bounce   - SIMULINK system modeling a
            bouncing ball.
 vdp      - SIMULINK system modeling the
            Van der Pol equations.
```

SIMULINK/Complex Systems.

```
 dblcart1 - SIMULINK system modeling a
            mass-spring system.
 dblpend1 - SIMULINK system modeling a
            double-pendulum system.
 dblpend2 - SIMULINK system modeling a
            double-pendulum system.
 penddemo - SIMULINK system modeling an
            inverted pendulum.
 dblcart  - SIMULINK system modeling a
            double-cart system.
 thermo   - SIMULINK system modeling a
            thermostat heating a house.
 f14      - SIMULINK system modeling an
            aircraft in flight.
```

SIMULINK/Advanced Products.

```
 xpaccel  - Provide information about the
            SIMULINK Accelerator.
 ccodegen - Provide information about the
            C-Code Generator.
```

Toolbox/Signal Processing.

```
 filtdem  - Signal Processing filter demo.
 filtdem2 - Demonstrate filter design
            techniques.
 sigdemo1 - Discrete-time Fourier transform
            of a signal.
 sigdemo2 - Continuous-time Fourier
            transform of a signal.
 phone    - Signal processing and the
            touch-tone phone.
```

Toolbox/System Identification.

```
 sysiddm  - Identify "hairdryer" system
            characteristics.
 iddems   - Set up System Identification
            command line demos.
```

Toolbox/Optimization.

```
 bandem   - Banana function minimization
            demonstration.
 optdems  - Set up Optimization command
            line demos.
```

Toolbox/Neural Networks.

```
 bckprp12 - Demonstrate backpropagation.
 bckprp62 - Demonstrate backpropagation
            with momentum.
 neural   - Neural network character
            recognition.
```

Toolbox/Control System.

```
 dskdemo  - Build controller for a disk
            read/write head.
 ctrldems - Set up Control System command
            line demos.
```

Toolbox/Robust Control.

```
 accdm2   - Demo of the 1990 ACC benchmark.
 rctdems  - Set up Robust Control command
            line demos.
```

Toolbox/Mu-Analysis and Synthesis.

```
 xpmu     - Description of the Mu-Analysis
            and Synthesis process.
 mudems   - Set up Mu-Analysis and
            Synthesis command line demos.
```

Toolbox/Spline.

```
 spapidm2 - Demonstrate spline
            interpolation.
 spldems  - Set up Spline command line
            demos.
```

Toolbox/Symbolic Math.

```
 xpcalc   - Calculus operations.
 xpgiv    - Givens transformation.
```

Toolbox/Image Processing.

```
 xpimage  - Demonstrate some Image
            Processing capabilities.
```

Toolbox/Statistics.

```
 xppolytl - Interactively fit a polynomial
            to noisy data.
 statdems - Set up Statistics command line
            demos.
```

Extras/Gallery.

```
 knot     - Tube surrounding a three-
```

```
            dimensional knot.
quivdemo - Demonstrate the quiver
            function.
modes    - Plot 12 modes of the L-shaped
            membrane.
logo     - Display the MATLAB L-shaped
            membrane logo.
klein1   - Construct a Klein bottle.
cruller  - Construct cruller.
tori4    - Construct four linked tori.
spharm2  - Construct spherical surface
            harmonic.
```

Extras/Games.

```
xpbombs  - Minesweeper game.
life     - Conway's Game of Life.
bblwrap  - Bubblewrap.
```

Extras/Miscellaneous.

```
truss    - Animation of a bending bridge
            truss.
travel   - Traveling salesman problem.
wrldtrv  - Great circle flight routes
            around the globe.
makevase - Generate and plot a surface of
            revolution.
logospin - Movie of The MathWorks' logo
            spinning.
crulspin - Spinning cruller movie.
xpquad   - Superquadrics plotting
            demonstration.
spinner  - Colorful lines spinning through
space.
```

Extras/Contact Info.

```
contact1 - How to reach The
            MathWorks, Inc.
contact2 - How to reach The MathWorks,
            Inc. by email.
contact3 - How to reach international
            agents for The MathWorks, Inc.
agents   - International distributors'
            locations and contact
            information.
```

14.8.4 Elfun (Elementary Math Functions)

Trigonometric.

```
sin      - Sine.
sinh     - Hyperbolic sine.
asin     - Inverse sine.
asinh    - Inverse hyperbolic sine.
cos      - Cosine.
cosh     - Hyperbolic cosine.
acos     - Inverse cosine.
acosh    - Inverse hyperbolic cosine.
tan      - Tangent.
tanh     - Hyperbolic tangent.
atan     - Inverse tangent.
atan2    - Four quadrant inverse tangent.
atanh    - Inverse hyperbolic tangent.
sec      - Secant.
sech     - Hyperbolic secant.
asec     - Inverse secant.
asech    - Inverse hyperbolic secant.
csc      - Cosecant.
csch     - Hyperbolic cosecant.
acsc     - Inverse cosecant.
acsch    - Inverse hyperbolic cosecant.
cot      - Cotangent.
coth     - Hyperbolic cotangent.
acot     - Inverse cotangent.
acoth    - Inverse hyperbolic cotangent.
```

Exponential.

```
exp      - Exponential.
log      - Natural logarithm.
log10    - Common logarithm.
sqrt     - Square root.
```

Complex.

```
abs      - Absolute value.
angle    - Phase angle.
conj     - Complex conjugate.
imag     - Complex imaginary part.
real     - Complex real part.
```

Numeric.

```
fix      - Round towards zero.
floor    - Round towards minus infinity.
ceil     - Round towards plus infinity.
round    - Round towards nearest integer.
rem      - Remainder after division.
sign     - Signum function.
```

14.8.5 Elmat (Elementary Matrices and Manipulation)

Elementary matrices.

```
zeros    - Zeros matrix.
ones     - Ones matrix.
eye      - Identity matrix.
rand     - Uniformly distributed random
            numbers.
randn    - Normally distributed random
            numbers.
linspace - Linearly spaced vector.
logspace - Logarithmically spaced vector.
meshgrid  - X and Y arrays for 3-D plots.
:         - Regularly spaced vector.
```

Special variables and constants.

```
ans      - Most recent answer.
eps      - Floating point relative
            accuracy.
realmax  - Largest floating point number.
realmin  - Smallest positive floating
            point number.
pi       - 3.1415926535897....
i, j     - Imaginary unit.
inf      - Infinity.
NaN      - Not-a-Number.
flops    - Count of floating point
            operations.
nargin   - Number of function input
            arguments.
nargout  - Number of function output
            arguments.
computer - Computer type.
isieee   - True for computers with IEEE
            arithmetic.
isstudent- True for the Student Edition.
why      - Succinct answer.
version  - MATLAB version number.
```

Time and dates.

```
clock    - Wall clock.
cputime  - Elapsed CPU time.
date     - Calendar.
etime    - Elapsed time function.
tic, toc - Stopwatch timer functions.
```

Matrix manipulation.

```
diag     - Create or extract diagonals.
fliplr   - Flip matrix in the left/right
            direction.
flipud   - Flip matrix in the up/down
            direction.
reshape  - Change size.
rot90    - Rotate matrix 90 degrees.
```

```
tril      - Extract lower triangular part.
triu      - Extract upper triangular part.
:         - Index into matrix, rearrange
            matrix.
```

14.8.6 Funfun (Function Functions)

Function functions - nonlinear numerical methods.

```
ode23     - Solve differential equations,
            low order method.
ode23p    - Solve and plot solutions.
ode45     - Solve differential equations,
            high order method.
quad      - Numerically evaluate integral,
            low order method.
quad8     - Numerically evaluate integral,
            high order method.
fmin      - Minimize function of one
            variable.
fmins     - Minimize function of several
            variables.
fzero     - Find zero of function of one
            variable.
fplot     - Plot function.
```

14.8.7 General (General Purpose Command)

Managing commands and functions.

```
help     - On-line documentation.
doc      - Load hypertext documentation.
what     - Directory listing of M-, MAT-
           and MEX-files.
type     - List M-file.
lookfor  - Keyword search through the HELP
           entries.
which    - Locate functions and files.
demo     - Run demos.
path     - Control MATLAB's search path.
```

Managing variables and the workspace.

```
who      - List current variables.
whos     - List current variables, long
           form.
load     - Retrieve variables from disk.
save     - Save workspace variables to
           disk.
clear    - Clear variables and functions
           from memory.
pack     - Consolidate workspace memory.
size     - Size of matrix.
length   - Length of vector.
disp     - Display matrix or text.
```

Working with files and the operating system.

```
cd       - Change current working directory.
dir      - Directory listing.
delete   - Delete file.
getenv   - Get environment value.
!        - Execute operating system command.
unix     - Execute operating system command
           & return result.
diary    - Save text of MATLAB session.
```

Controlling the command window.

```
cedit    - Set command line edit/recall
           facility parameters.
clc      - Clear command window.
home     - Send cursor home.
format   - Set output format.
echo     - Echo commands inside script
           files.
more     - Control paged output in
           command window.
```

Starting and quitting from MATLAB.

```
quit      - Terminate MATLAB.
startup   - M-file executed when MATLAB is
            invoked.
matlabrc  - Master startup M-file.
```

General information.

```
info      - Information about MATLAB and
            The MathWorks, Inc.
subscribe - Become subscribing user of
            MATLAB.
hostid    - MATLAB server host
            identification number.
whatsnew  - Information about new features
            not yet documented.
ver       - MATLAB, SIMULINK, and TOOLBOX
            version information.
```

14.8.8 Graphics (General Purpose Graphics Functions)

Figure window creation and control.

```
figure   - Create Figure (graph window).
gcf      - Get handle to current figure.
clf      - Clear current figure.
close    - Close figure.
```

Axis creation and control.

```
subplot - Create axes in tiled positions.
axes     - Create axes in arbitrary
           positions.
gca      - Get handle to current axes.
cla      - Clear current axes.
axis     - Control axis scaling and
           appearance.
caxis    - Control pseudocolor axis
           scaling.
hold     - Hold current graph.
```

Handle Graphics objects.

```
figure    - Create figure window.
axes      - Create axes.
line      - Create line.
text      - Create text.
patch     - Create patch.
surface   - Create surface.
image     - Create image.
uicontrol- Create user interface control.
uimenu    - Create user interface menu.
```

Handle Graphics operations.

```
set       - Set object properties.
get       - Get object properties.
reset     - Reset object properties.
delete    - Delete object.
gco       - Get handle to current object.
drawnow   - Flush pending graphics events.
newplot   - M-file preamble for NextPlot
            property.
findobj   - Find objects with specified
            property values.
```

Hardcopy and storage.

```
print    - Print graph or save graph to
           file.
printopt- Configure local printer
           defaults.
orient   - Set paper orientation.
capture  - Screen capture of current
           figure.
```

Movies and animation.

```
moviein - Initialize movie frame memory.
getframe- Get movie frame.
movie   - Play recorded movie frames.
```

Miscellaneous.

```
ginput    - Graphical input from mouse.
ishold    - Return hold state.
graymon   - Set graphics window defaults
            for gray-scale monitors.
rbbox     - Rubberband box.
rotate    - Rotate an object about a
            specified direction.
terminal  - Set graphics terminal type.
uiputfile- Put up dialog box for saving
            files.
uigetfile- Put up dialog box which queries
            for file names.
whitebg   - Set graphics window defaults
            for white background.
zoom      - Zoom in and out on a 2-D plot.
waitforbuttonpress- Wait for
            key/buttonpress over figure.
```

14.8.9 Iofun (Low-Level File I/O Functions)

File opening and closing.

```
fopen    - Open file.
 fclose  - Close file.
```

Unformatted I/O.

```
fread    - Read binary data from file.
fwrite   - Write binary data to file.
```

Formatted I/O.

```
fscanf   - Read formatted data from file.
fprintf  - Write formatted data to file.
fgetl    - Read line from file, discard
           newline character.
fgets    - Read line from file, keep
           newline character.
```

File positioning.

```
ferror   - Inquire file I/O error status.
feof     - Test for end-of-file.
fseek    - Set file position indicator.
ftell    - Get file position indicator.
frewind  - Rewind file.
```

String conversion.

```
sprintf  - Write formatted data to string.
sscanf   - Read string under format
           control.
```

File Import/Export Routines.

WK1 Format.

```
wk1const - WK1 record definitions.
wk1read  - Read WK1 file/range.
wk1write - Write out matrix in a WK1
           formatted file.
wk1wrec  - Write a WK1 record header.
```

CSV Format.

```
csvread  - Read Comma Separated Value
           formatted file into a matrix.
csvwrite - Write out matrix in a CSV
           formatted file.
```

ASCII Delimited Format.

```
dlmread  - Read ASCII delimited file into
           a matrix.
dlmwrite - Write out matrix in ASCII
           delimited file format.
```

14.8.10 Lang (Language Constructs and Debuggings)

MATLAB as a programming language.

```
script   - About MATLAB scripts and M-
           files.
function - Add new function.
eval     - Execute string with MATLAB
           expression.
feval    - Execute function specified by
           string.
global   - Define global variable.
nargchk  - Validate number of input
           arguments.
lasterr  - Last error message.
```

Control flow.

```
if      - Conditionally execute statements.
else    - Used with IF.
elseif  - Used with IF.
end     - Terminate the scope of FOR, WHILE
          and IF statements.
for     - Repeat statements a specific
          number of times.
while   - Repeat statements an indefinite
          number of times.
break   - Terminate execution of loop.
return  - Return to invoking function.
error   - Display message and abort
          function.
```

Interactive input.

```
input     - Prompt for user input.
keyboard  - Invoke keyboard as if it were
            a Script-file.
menu      - Generate menu of choices for
            user input.
pause     - Wait for user response.
uimenu    - Create user interface menu.
uicontrol - Create user interface control.
```

Debugging commands.

```
dbstop   - Set breakpoint.
dbclear  - Remove breakpoint.
dbcont   - Resume execution.
dbdown   - Change local workspace context.
dbstack  - List who called whom.
dbstatus - List all breakpoints.
dbstep   - Execute one or more lines.
dbtype   - List M-file with line numbers.
dbup     - Change local workspace context.
dbquit   - Quit debug mode.
mexdebug - Debug MEX-files.
```

14.8.11 Matfun (Matrix Functions)

Matrix analysis.

```
cond     - Matrix condition number.
norm     - Matrix or vector norm.
rcond    - LINPACK reciprocal condition
           estimator.
rank     - Number of linearly independent
           rows or columns.
det      - Determinant.
trace    - Sum of diagonal elements.
null     - Null space.
orth     - Orthogonalization.
rref     - Reduced row echelon form.
```

Linear equations.

```
\ and /  - Linear equation solution; use
```

```
            "help slash".
 chol      - Cholesky factorization.
 lu        - Factors from Gaussian
             elimination.
 inv       - Matrix inverse.
 qr        - Orthogonal-triangular
             decomposition.
 qrdelete - Delete a column from the QR
             factorization.
 qrinsert - Insert a column in the QR
             factorization.
 nnls      - Non-negative least-squares.
 pinv      - Pseudoinverse.
 lscov     - Least squares in the presence
             of known covariance.
```

Eigenvalues and singular values.

```
 eig       - Eigenvalues and
             eigenvectors.
 poly      - Characteristic polynomial.
 polyeig   - Polynomial eigenvalue problem.
 hess      - Hessenberg form.
 qz        - Generalized eigenvalues.
 rsf2csf   - Real block diagonal form to
             complex diagonal form.
 cdf2rdf   - Complex diagonal form to real
             block diagonal form.
 schur     - Schur decomposition.
 balance   - Diagonal scaling to improve
             eigenvalue accuracy.
 svd       - Singular value decomposition.
```

Matrix functions.

```
 expm      - Matrix exponential.
 expm1     - M-file implementation of expm.
 expm2     - Matrix exponential via Taylor
             series.
 expm3     - Matrix exponential via
             eigenvalues and eigenvectors.
 logm      - Matrix logarithm.
 sqrtm     - Matrix square root.
 funm      - Evaluate general matrix
             function.
```

14.8.12 OPS (Operators and Special Characters)

Arithmetic and Matrix Operators.

```
Char     Name                          HELP topic

 +       Plus                            arith
 -       Minus                           arith
 *       Matrix multiplication           arith
 .*      Array multiplication            arith
 ^       Matrix power                    arith
 .^      Array power                     arith

 \       Backslash or left division      slash
 /       Slash or right division         slash
 ./      Array division                  slash
 kron    Kronecker tensor product        kron

 :       Colon                           colon

 ( )     Parentheses                     paren
 [ ]     Brackets                        paren

 .       Decimal point                   punct
 ..      Parent directory                punct
 ...     Continuation                    punct
 ,       Comma                           punct
 ;       Semicolon                       punct
 %       Comment                         punct
 !       Exclamation point               punct
 '       Transpose and quote             punct

 =       Assignment                      punct

 ==      Equality                        relop
 < >     Relational operators            relop
 &       Logical AND                     relop
 |       Logical OR                      relop
 ~       Logical NOT                     relop
 xor     Logical EXCLUSIVE OR            xor
```

Logical characteristics.

```
 exist     - Check if variables or functions
             are defined.
 any       - True if any element of vector
             is true.
 all       - True if all elements of vector
             are true.
 find      - Find indices of non-zero
             elements.
 isnan     - True for Not-A-Number.
 isinf     - True for infinite elements.
 finite    - True for finite elements.
 isempty   - True for empty matrix.
 isreal    - True for real matrix.
 issparse - True for sparse matrix.
 isstr     - True for text string.
 isglobal - True for global variables.
```

14.8.13 Plotxy (Two-Dimensional Graphics)

Elementary X-Y graphs.

```
 plot      - Linear plot.
 loglog    - Log-log scale plot.
 semilogx- Semi-log scale plot.
 semilogy- Semi-log scale plot.
 fill      - Draw filled 2-D polygons.
```

Specialized X-Y graphs.

```
 polar     - Polar coordinate plot.
 bar       - Bar graph.
 stem      - Discrete sequence or "stem"
             plot.
 stairs    - Stairstep plot.
 errorbar- Error bar plot.
 hist      - Histogram plot.
 rose      - Angle histogram plot.
 compass  - Compass plot.
 feather  - Feather plot.
 fplot     - Plot function.
 comet     - Comet-like trajectory.
```

Graph annotation.

```
 title     - Graph title.
 xlabel    - X-axis label.
 ylabel    - Y-axis label.
 text      - Text annotation.
 gtext     - Mouse placement of text.
 grid      - Grid lines.
```

14.8.14 Plotxyz (Three-Dimensional Graphics)

Line and area fill commands.

```
 plot3     - Plot lines and points in 3-D
             space.
 fill3     - Draw filled 3-D polygons in 3-D
             space.
 comet3    - 3-D comet-like trajectories.
```

Contour and other 2-D plots of 3-D data.

```
 contour   - Contour plot.
 contour3 - 3-D contour plot.
 clabel    - Contour plot elevation labels.
 contourc - Contour plot computation (used
             by contour).
```

```
pcolor    - Pseudocolor (checkerboard)
            plot.
quiver    - Quiver plot.
```

Surface and mesh plots.

```
mesh      - 3-D mesh surface.
meshc     - Combination mesh/contour plot.
meshz     - 3-D Mesh with zero plane.
surf      - 3-D shaded surface.
surfc     - Combination surf/contour plot.
surfl     - 3-D shaded surface with
            lighting.
waterfall- Waterfall plot.
```

Volume visualization.

```
slice     - Volumetric visualization plots.
```

Graph appearance.

```
view      - 3-D graph viewpoint
            specification.
viewmtx   - View transformation matrices.
hidden    - Mesh hidden line removal mode.
shading   - Color shading mode.
axis      - Axis scaling and appearance.
caxis     - Pseudocolor axis scaling.
colormap - Color look-up table.
```

Graph annotation.

```
title     - Graph title.
xlabel    - X-axis label.
ylabel    - Y-axis label.
zlabel    - Z-axis label for 3-D plots.
text      - Text annotation.
gtext     - Mouse placement of text.
grid      - Grid lines.
```

3-D objects.

```
cylinder - Generate cylinder.
sphere    - Generate sphere.
```

14.8.15 Polyfun (Polynomial and Interpolation Functions)

Polynomials.

```
roots     - Find polynomial roots.
poly      - Construct polynomial with
            specified roots.
polyval   - Evaluate polynomial.
polyvalm - Evaluate polynomial with matrix
            argument.
residue   - Partial-fraction expansion
            (residues).
polyfit   - Fit polynomial to data.
polyder   - Differentiate polynomial.
conv      - Multiply polynomials.
deconv    - Divide polynomials.
```

Data interpolation.

```
interp1   - 1-D interpolation (1-D table
            lookup).
interp2   - 2-D interpolation (2-D table
            lookup).
interpft - 1-D interpolation using FFT
            method.
griddata - Data gridding.
```

Spline interpolation.

```
spline    - Cubic spline data
            interpolation.
ppval     - Evaluate piecewise polynomial.
```

14.8.16 Sparfun (Sparce Matrix Functions)

Elementary sparse matrices.

```
speye     - Sparse identity matrix.
sprandn   - Sparse random matrix.
sprandsym- Sparse symmetric random matrix.
spdiags   - Sparse matrix formed from
            diagonals.
```

Full to sparse conversion.

```
sparse    - Create sparse matrix from
            nonzeros and indices.
full      - Convert sparse matrix to full
            matrix.
find      - Find indices of nonzero
            entries.
spconvert- Convert from sparse matrix
            external format.
```

Working with nonzero entries of sparse matrices.

```
nnz       - Number of nonzero entries.
nonzeros - Nonzero entries.
nzmax     - Amount of storage allocated for
            nonzero entries.
spones    - Replace nonzero entries with
            ones.
spalloc   - Allocate memory for nonzero
            entries.
issparse - True if matrix is sparse.
spfun     - Apply function to nonzero
            entries.
```

Visualizing sparse matrices.

```
spy       - Visualize sparsity structure.
gplot     - Plot graph, as in "graph
            theory".
```

Reordering algorithms.

```
colmmd    - Column minimum degree.
symmmd    - Symmetric minimum degree.
symrcm    - Reverse Cuthill-McKee ordering.
colperm   - Order columns based on nonzero
            count.
randperm - Random permutation vector.
dmperm    - Dulmage-Mendelsohn
            decomposition.
```

Norm, condition number, and rank.

```
normest   - Estimate 2-norm.
condest   - Estimate 1-norm condition.
sprank    - Structural rank.
```

Operations on trees.

```
treelayout- Lay out a tree or forest.
treeplot  - Plot a picture of a tree.
etree      - Elimination tree of a matrix.
etreeplot - Plot the elimination tree.
```

Miscellaneous.

```
symbfact  - Symbolic factorization
            analysis.
spparms   - Set parameters for sparse
            matrix routines.
spaugment- Form least squares augmented
            system.
```

14.8.17 Specfun (Specialized Math Function)

```
besselj   - Bessel function of the first
            kind.
bessely   - Bessel function of the second
            kind.
besseli   - Modified Bessel function of the
            first kind.
```

```
besselk  - Modified Bessel function of the
           second kind.
beta     - Beta function.
betainc  - Incomplete beta function.
betaln   - Logarithm of beta function.
ellipj   - Jacobi elliptic functions.
ellipke  - Complete elliptic integral.
erf      - Error function.
erfc     - Complementary error function.
erfcx    - Scaled complementary error
           function.
erfinv   - Inverse error function.
expint   - Exponential integral function.
gamma    - Gamma function.
gcd      - Greatest common divisor.
gammainc - Incomplete gamma function.
lcm      - Least common multiple.
legendre - Associated Legendre function.
gammaln  - Logarithm of gamma function.
log2     - Dissect floating point numbers.
pow2     - Scale floating point numbers.
rat      - Rational approximation.
rats     - Rational output.
cart2sph - Transform from Cartesian to
           spherical coordinates.
cart2pol - Transform from Cartesian to
           polar coordinates.
pol2cart - Transform from polar to
           Cartesian coordinates.
sph2cart - Transform from spherical to
           Cartesian coordinates.
```

14.8.18 Specmat (Specialized Matrices)

```
compan   - Companion matrix.
gallery  - Several small test matrices.
hadamard - Hadamard matrix.
hankel   - Hankel matrix.
hilb     - Hilbert matrix.
invhilb  - Inverse Hilbert matrix.
kron     - Kronecker tensor product.
magic    - Magic square.
pascal   - Pascal matrix.
rosser   - Classic symmetric eigenvalue
           test problem.
toeplitz - Toeplitz matrix.
vander   - Vandermonde matrix.
wilkinson- Wilkinson's eigenvalue test
           matrix
```

14.8.19 Sounds (Sound Processing Functions)

General sound functions.

```
sound    - Convert vector into sound.
saxis    - Sound axis scaling.
```

Computer-specific sound functions.

```
auwrite  - Write mu-law encloded audio
           file.
auread   - Read mu-law encloded audio
           file.
wavwrite - Write MS Windows .WAV audio
           file.
wavread  - Read MS Windows .WAV audio
           file.
mu2lin   - Mu-law to linear conversion.
lin2mu   - Linear to mu-law conversion.
```

14.8.20 Strfun (Character String Functions)

General.

```
strings  - About character strings in
           MATLAB.
abs      - Convert string to numeric
           values.
setstr   - Convert numeric values to
           string.
isstr    - True for string.
blanks   - String of blanks.
deblank  - Remove trailing blanks.
str2mat  - Form text matrix from
           individual strings.
eval     - Execute string with MATLAB
           expression.
```

String comparison.

```
strcmp   - Compare strings.
findstr  - Find one string within another.
upper    - Convert string to uppercase.
lower    - Convert string to lowercase.
isletter - True for letters of the
           alphabet.
isspace  - True for white space
           characters.
strrep   - Replace a string with another.
strtok   - Find a token in a string.
```

String to number conversion.

```
num2str  - Convert number to string.
int2str  - Convert integer to string.
str2num  - Convert string to number.
sprintf  - Convert number to string under
           format control.
sscanf   - Convert string to number under
           format control.
```

Hexadecimal to number conversion.

```
hex2num  - Convert hex string to IEEE
           floating point number.
hex2dec  - Convert hex string to decimal
           integer.
dec2hex  - Convert decimal integer to hex
           string.
```