Corrective Methods: Implementation
TABLE OF CONTENTS

§27.1 Introduction .............................................. 27–3
§27.2 The Corrective Phase ...................................... 27–3
§27.3 Solving the Newton Systems .............................. 27–4
  §27.3.1 Newton Iteration for Load Control ................. 27–4
  §27.3.2 Newton Iteration for State Control ................. 27–4
  §27.3.3 Newton Iteration for Arclength Control ............ 27–5
  §27.3.4 Newton Iteration for External Work Control ....... 27–5
§27.4 Organization of GeNoBeC ................................. 27–6
  §27.4.1 Nonlinear Solution Driver ......................... 27–6
  §27.4.2 Correction Driver Module ......................... 27–7
  §27.4.3 Iteration Cycle Module ......................... 27–8
§27.5 Sample Scripts .............................................. 27–10
  §27.5.1 Mises Truss Template Script ................... 27–10
  §27.5.2 Circle Game Template Script .................. 27–10
  §27.5.3 Circle Game Variant Template Script .......... 27–10
§27. Exercises .................................................. 27–11


§27.2 The Corrective Phase

§27.1. Introduction

§27.2. The Corrective Phase

For convenience we recapitulate below material taken from ? in a previous Chapter.

All computations that follow pertain to the \((n + 1)^{th}\) incremental step. Hence for simplicity we shall omit the step subscript from all ensuing formulas. Starting from the predicted approximation \((u_0^n, \lambda_0^n)\), a Newton-like method applied to the algebraic system \((?\) generates a sequence of iterates

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

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

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

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

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

we find that the corrections

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

can be obtained by solving the linear algebraic system

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

Here all known quantities are evaluated at \(\boldsymbol{u}^k\) and \(\lambda^k\), for example \(\boldsymbol{r}^k = \boldsymbol{r}(\boldsymbol{u}^k, \lambda^k)\). For notational simplicity, however, the \(k\) superscript will kept out of \(\boldsymbol{d}, \eta, \boldsymbol{r}, \boldsymbol{K}, \boldsymbol{q}, \boldsymbol{a},\) and \(\boldsymbol{g}\), unless it is desirable to make the dependency on the iteration index explicit. If the tangent stiffness matrix \(\boldsymbol{K}\) is of order \(N\), the coefficient matrix of the linear system (27.6) has order \(N + 1\). This matrix is called the augmented stiffness matrix.

Note that although generally the tangent stiffness \(\boldsymbol{K}\) is symmetric and sparse, the augmented stiffness is generally unsymmetric (but see Exercise 27.2). Furthermore, its sparseness may be detrimentally
affected by the augmentation if $a^T$ links all DOF. It is therefore of interest to treat the linear system (27.6) with techniques that preserve both symmetry and sparseness.

The solution procedures described below make use of auxiliary systems of equations to achieve that goal. The number of auxiliary systems depends on whether the tangent stiffness $K$ is nonsingular (regular points) or singular (critical points). For the latter we have to distinguish between limit points and bifurcation points. In the present Chapter we shall concentrate on the treatment of regular points.

§27.3. Solving the Newton Systems

Recall from Chapter 4 that regular points of (25.1) are equilibrium solutions $(u, \lambda)$ at which the tangent stiffness matrix $K = \partial r/\partial u$ is nonsingular. If this property holds at the current solution iterate, we can perform forward Gauss elimination on (27.6) to get rid of $d$ and produce the following scalar equation for $\eta$:

$$ (g + a^T K^{-1} q) \eta = -c + a^T K^{-1} r. \quad \text{(27.7)} $$

Let $v_r$ and $v_q$ denote the solution of the symmetric linear systems

$$ K v_r = -r, \quad K v_q = q. \quad \text{(27.8)} $$

Here $v_q$ is our old friend the incremental velocity vector $v = K^{-1}$, whereas $v_r = -K^{-1} r$ is called the residual velocity vector. (Subscripts are appended to distinguish the right-hand sides). Using (27.8) and solving for $\eta$ from (27.7), the solution of (27.6) can be compactly expressed as

$$ \eta = -c + a^T v_r, \quad d = v_r + \eta v_q. \quad \text{(27.9)} $$

It is seen that two right hand sides, $r$ and $q$, have to be generally solved for at each Newton step. The number reduces to one for $k > 1$, however, if (i) modified Newton is used so that $K$ is held fixed for several steps and (ii) $q$ does not vary. The second assumption holds in structural mechanics applications if the loading is conservative and proportional. (The modified Newton method is described in Chapter 23.)

We now specialize the Newton iteration to popular increment control strategies. In the following subsections, the incremental step number $n$ and the iteration index $k$ are restored for clarity.

§27.3.1. Newton Iteration for Load Control

For load control, also called $\lambda$ control, $c = \Delta \lambda_n - \ell_n = 0$, whence $a^T = \partial c/\partial u = 0$ and $g = \partial c/\partial \lambda = 1$. Thus

$$ \eta^k = -c^k, \quad d^k = v_r^k - c^k v_q^k. \quad \text{(27.10)} $$

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

$$ \lambda_{n+1}^k = \lambda_n^k = \lambda_n + \ell_n, \quad u_{n+1}^k = u_{n+1}^k + v_r^k = u_{n+1}^k - (K^k)^{-1} r_k. \quad \text{(27.11)} $$

In this case $\lambda_{n+1}^k$ stays fixed for all $k$, and only the solution $v_r^k$ for the residual RHS is needed. This strategy fails at critical points.

1 Another way to get (27.7) is to solve for $d$ from the first equation: $d = K^{-1} (q \eta - r)$, and replace $d$ into the second one: $A^T d + g \eta = c$. 

27–4
§27.3.2. Newton Iteration for State Control

For unscaled state control the constraint is
\[ c = (\Delta u_n)^T (\Delta u_n) - l_n^2 = 0. \] (27.12)

Thus \( a^T = \partial c / \partial u = 2 \Delta u_n \) and \( g = \partial c / \partial \lambda = 0 \). Substituting into (27.9) gives
\[ \eta^k = -\frac{c^k + 2(\Delta u_n^k)^T v^k_r}{2(\Delta u_n^k)^T v^k_q}, \quad d^k = v^k_r + \eta^k v^k_q. \] (27.13)

If all solution iterates satisfy the constraint \( c = 0 \), (27.13) reduces to
\[ \eta^k = -\frac{(\Delta u_n^k)^T v^k_r}{(\Delta u_n^k)^T v^k_q}, \quad d^k = v^k_r + \eta^k v^k_q. \] (27.14)

The corrective process becomes
\[ \lambda_{n+1}^k = \lambda_n^k + \eta^k, \quad u_{n+1}^k = u_n^k + \eta^k d^k = u_n^k + v^k_r + \eta^k v^k_q. \] (27.15)

For the simplest scaled state control: \( c = (\Delta u_n)^T (\Delta u_n) / L_{ref}^2 - l_n^2 = 0 \), in which \( L_{ref} \) is a reference length (kept constant during the iterations), the modifications are obvious.

Evidently the first of (27.14) “blows up” if either \( \Delta u_n^k = 0 \), \( v^k_r = 0 \), or \( \Delta u_n^k \) and \( v^k_q \) are orthogonal. The first scenario occurs at turning points where all displacements “freeze.” The second one occurs when the incremental load vanishes. The last one occurs by definition at bifurcation points. Those failure possibilities are rarely pointed out by the advocates of that strategy.

§27.3.3. Newton Iteration for Arclength Control

For unscaled arclength control the constraint \( c = 0 \) is given as (27.16), reproduced for convenience:
\[ c = \frac{1}{f_n} (v_n^T \Delta u_n + \Delta \lambda_n) - l_n = 0, \quad \text{in which} \quad f_n = \pm \sqrt{1 + v_n^T v_n}. \] (27.16)

In this case \( a = \partial c / \partial u = v_n / f_n \) and \( g = \partial c / \partial \lambda = 1 / f_n \), in which \( f_n = \sqrt{1 + v_n^T v_n} \). Substituting into (27.9) gives
\[ \eta^k = -\frac{c^k + (1/f_n)(v_n^k)^T v^k_r}{(1/f_n)(1 + (v_n^k)^T v^k_q)}, \quad d^k = v^k_r + \eta^k v^k_q. \] (27.17)

If all solution iterates satisfy the constraint \( c = 0 \), the factor \( 1/f_n \) cancels out, reducing (27.17) to
\[ \eta^k = -\frac{(v_n^k)^T v^k_r}{1 + (v_n^k)^T v^k_q}, \quad d^k = v^k_r + \eta^k v^k_q. \] (27.18)

The corrective process becomes
\[ \lambda_{n+1}^k = \lambda_n^k + \eta^k, \quad u_{n+1}^k = u_n^k + \eta^k d^k = u_n^k + v^k_r + \eta^k v^k_q. \] (27.19)

At limit points \( ||v^k|| \to \infty \), forcing \( \eta^k \to 0 \) and \( d^k \to v^k_r \), which is the correct behavior.

For the simplest scaled arclength control \( v_n \) is replaced by \( \tilde{v}_n = v_n / L_{ref} \), where \( L_{ref} \) is a reference length that is kept constant during the iterations. The ensuing modifications are straightforward. More complicated scalings may specify a diagonal matrix \( S \) of one-over-length scaling factors, in which case the modifications involve the quadratic form \( v^T S^2 d \).
§27.3.4. Newton Iteration for External Work Control

For external work control the constraint \( c = 0 \) is

\[
c = q_n^T \Delta u_n - \ell_n = 0, \tag{27.20}
\]

in which \( q_n \) is kept constant during the iterations, even if the load is nonproportional. Thus \( a^T = \partial c / \partial u = q_n \) and \( g = \partial c / \partial \lambda = 0 \). If all solution iterates satisfy the constraint \( c = 0 \), we get

\[
\eta^k = -\left( \frac{q_n^k}{q_n^r} \right)^T v_r^k, \quad d^k = v_r^k + \eta^k v_q^k. \tag{27.21}
\]

The corrective process becomes

\[
\lambda_{n+1}^k = \lambda_{n+1}^k + \eta^k, \quad u_{n+1}^k = u_{n+1}^k + \eta^k d^k = u_{n+1}^k + v_r^k + \eta^k v_q^k. \tag{27.22}
\]

The process runs into trouble if \( q \) vanishes since then \( \eta^k \to \infty \).

§27.4. Organization of GeNoBeC

The general organization of the GeNoBeC program follows the schematics of Figure 21.2. the names of the modules are listed in Figure 27.1.

Three modules have been added to upgrade the incremental solution process with corrections: CorrStep, IterCycle and IncResVel, the names of which are shown in blue. In addition, the code of the nonlinear solution driver NonLinSolDriver is slightly expanded to call CorrStep and the correction parameter argument activated. Those four new or updated modules are described in the following subsections.
§27.4 ORGANIZATION OF GENOBEC

The code of the nonlinear solving module NonLinSolver is listed in Figure 27.2. It is invoked as

```
{soltab, status} = NonLinSolver[problem, method, geopar, matpar, fabpar, incpar, conpar, refct, refsta, rflode, options];
```

This has exactly the same argument sequence of the eponymous module for GeNoMe, as described in §27.4.1. Two arguments, however, contain additional information:

- **method**
  The integration method identifier, as listed in Table 27.2.

- **ics**
  The increment control strategy identifier, as listed in Table 27.2.

- **cmi**
  The correction method identifier. This is the new item, which is listed in Table 27.2.

- **corpar**
  A list of control parameters. See Table 27.2.

The function returns are the same as in GeNoBe: the computed solution table in soltab and the exit status in status. The configuration of the solution table is slightly different, however, because item # 10 in each solution row records information gathered in the corrective iteration in each step. See Table 27.2.

The additional code in NonLinSolver is a call to CorStep in the step-by-step While loop that advances the solution. See Figure 27.2.

§27.4.2. Correction Driver Module

Module CorIterDriver, listed in Figure 27.2, drives the corrective iteration process. It is invoked as

```
{nextsol, status} = CorIterDriver[problem, method, geopar, matpar, fabpar, incpar, conpar, refct, refsta, rflode, options];
```
Chapter 27: CORRECTIVE METHODS: IMPLEMENTATION

IncStep[problem_, method_, geopar_, matpar_, fabpar_, incpar_, corpar_,
ctlpar_, state_, rfload_, options_, soldim_]:= Module[
{int, ics, A, c, w, neval, nextsol, status=""},
{int, ics}=method;
{A, c, w}=ButcherTable[int]; neval=Length[c];
If [neval==1,
{nextsol, status}=IncStep1[problem, method, geopar, matpar, fabpar,
incpar, corpar, ctlpar, state, rfload, options, soldim];
If [neval==2,
{nextsol, status}=IncStep2[problem, method, geopar, matpar, fabpar,
incpar, corpar, ctlpar, state, rfload, options, soldim];
If [neval==3,
{nextsol, status}=IncStep3[problem, method, geopar, matpar, fabpar,
incpar, corpar, ctlpar, state, rfload, options, soldim];
If [neval==4,
{nextsol, status}=IncStep4[problem, method, geopar, matpar, fabpar,
incpar, corpar, ctlpar, state, rfload, options, soldim];
Return[{nextsol, status}];

Figure 27.3. The CorDriver module GeNoMeC that performs the corrective iteration process.

IncVel[problem_, method_, geopar_, matpar_, fabpar_, incpar_,
corpar_, ctlpar_, state_, rfload_, options_]:= Module[
{probid=problem[[1]], prbvar="", K, Kdet, Knorm, Knill, Kcinv,
ev, evmin, evmax, epsill=10.^[(-6)], epsing=10.^[(-12)],
numer=options[[1]], status="", Null3=Table[Null,{3}], q, v},
If [Length[problem]>1, prbvar=problem[[2]]];
K=TanStiff[problem, method, geopar, matpar, fabpar, incpar,
corpar, ctlpar, state, rfload, options];
If [K==Null, status="K evaluation error"; Return[{Null3, status}];
Kdet=Det[K];
If ![numer, Kdet=Simplify[Kdet];
If [Kdet==0, status="Singular K"; Return[{Null3, status}]]; Knorm=MtxFrobNorm[K];
if [Knorm==0, status="Singular K"; Return[{Null3, status}];
Knill=epsill*Knorm;
evmax=Max[Max[ev], epsing]; evmin=Min[ev];
Kcinv=evmin/evmax; status="Ill conditioned K";
If [Kcinv<=epsing, status="Singular K"; Return[{Null3, status}]];];
q=IncLoad[problem, method, geopar, matpar, fabpar, incpar, corpar,
ctlpar, state, rfload, options];
v=LinearSolve[K, q]; If ![numer, v=Simplify[v]]; ClearAll[K];
Return[{v, q, Kcinv, status}];

Figure 27.4. The CorCycle module GeNoMeC that advances the solution over one cycle of
the corrective iteration process.

Most of the arguments of this module have already been described. Exceptions are
predsol The solution table row predicted by the incremental step driver module IncStep.
This actually nextsol returned by IncStep. It contains the predicted control parameter \(\lambda_n\) and state vector \(ubold_n\) from which CorIterDriver starts the iterative process.
The module returns nextsol and status, which are the corrected solution and exit status, respectively.

§27.4.3. Iteration Cycle Module

Module CorIterCycle, listed in Figure 27, performs one iteration (also called cycle) of the corrective

27–8
process. It is invoked as

\[
\{\text{nextsol}, \text{status}\} = \text{CorCycleCycle}[\text{problem}, \text{method}, \text{geopar}, \text{matpar}, \text{fabpar}, \\
\text{incpar}, \text{conpar}, \text{refctl}, \text{refsta}, \text{rfload}, \text{options}] ;
\]

The arguments of this module have already been described.
The module returns nextsol and status, which are the advanced control and state solutions, respectively.
§27.5. Sample Scripts

The ensuing scripts that run benchmark problems are provided as part of the *GeNoMeC* Notebook. All of them are provided in the posted code, and may be used as “templates” to start the assigned homework exercises of this Chapter.

§27.5.1. Mises Truss Template Script

The script listed in Figure 27.5 runs the Mises Truss (MT) benchmark problem. The reference configuration is $\lambda = u_X = u_Y = 0$. The script uses a specific combination of integration scheme (Forward Euler) and increment control strategy (Load Control), and displays results. Refer to the interspersed comments for the setting up details.

§27.5.2. Circle Game Template Script

The script listed in Figure 27.6 runs the Circle Game (CG) circle-only benchmark problem, identified as the "$\ldots\ldots$ variant. The reference configuration is $\lambda = \mu = 1/\sqrt{2}$, which lies on the circle first quadrant. The script uses a specific combination of integration scheme (Forward Euler) and increment control strategy (Load Control), and displays results. Refer to the interspersed comments for the setting up details.

§27.5.3. Circle Game Variant Template Script

The script listed in Figure 27.7 runs the Circle Game (CG) circle-plus-straight-line benchmark problem, identified as the "BB" variant. The reference configuration is $\lambda = \mu = 0$. The script uses a specific combination of integration scheme (Forward Euler) and increment control strategy (Load Control), and displays results. Refer to the interspersed comments for the setting up details.
ClearAll[S, H, Em, A0, nmax, λmax, umax, ell, adapt, ell, ell, acctol, cscale];

(* Set tags: benchmark-problem, problem-variant (blank here) *)
advancing-integration-scheme, increment-control-strategy.
Internally: problem={probid,prbvar} & method={int,ics};
N.B.: ics links w/ cscale, declared {} for default *)
problem={"MT", " "}; method={"FE","LC"}; cscale={};

(* Define structural properties: {span S, crown height H, elastic modulus Em, x-sec A0 of 2 arch members, no 3rd member *)
S=2; H=1; Em=10; A0=1;
geopar={S,H}; matpar={Em}; fabpar={A0,A0,0};

(* Ext force is λ (reference load: λ*rfload=λ*{0,-Em*A0} *)
rfload={0,-Em*A0};

(* Define incremental sol parameters - see above re cscale *)
nmax=55; λmax=2.; umax=N[2*H]; ell=0.04; adapt=False; acctol=0;
incpar={nmax, λmax, umax, ell, adapt, ell, ell, acctol, cscale};

(* Correction parameters are not used - set to empty list *)
corpar={};

(* Specify numerical (floating-point) computation *)
numer=True; options={numer};

(* Specify ref (initial) configuration as λ=0 and uX=uY=0 *)
refctl={0}; refsta={0,0};

(* Carry out nonlinear solution, which returns in soltab *)
{soltab,status}=NonLinSolDriver[problem,method,geopar,matpar, fabpar,incpar,corpar,refctl,refsta,rfload,options];
If [status=" " ,Print[status]]; (* Print solution table *)
doflab={"uX","uY"}; digits={{5,3}};
PrintSolTable[problem,method,soltab,doflab,digits];

(* Show selective response plots *)
aspect=0.6; (* aspect ratio *) imgsiz=350; (* width in pts *)
xrange=yrange={}; (* for future use *)
PlotSolTable[problem,method,soltab,{"uY","",doflab, xrange,yrange,aspect,imgsiz,"Black",False}];
PlotSolTable[problem,method,soltab,{"uX","",doflab, xrange,yrange,aspect,imgsiz,"Black",False}];
PlotSolTable[problem,method,soltab,{"step","uX"},doflab, xrange,yrange,aspect,imgsiz,"Black",False];
PlotSolTable[problem,method,soltab,{"||r||","uY"},doflab, xrange,yrange,aspect,imgsiz,"Red",False];
PlotSolTable[problem,method,soltab,{"step","Kdet"},doflab, xrange,yrange,aspect,imgsiz,"Blue",False];

\textbf{Figure} 27.5. Script “template” to run the Mises truss benchmark problem.
ClearAll[nmax, \(\lambda\)max, umax, ell, adapt, ell, ell, acctol, cscale];

(* Set tags: benchmark-problem, problem-variant (blank here)   
  advancing-integration-scheme, increment-control-strategy.   
  Internally: problem={probid,prbvar} & method={int,ics};   
  N.B.: ics links w/ cscale, declared {} for default *)

problem={"CG"," "}; method={"FE","LC"}; cscale={};

(* No definition of structural properties: needed.   
  Set geometry, material and fabrication parameters to empty lists *)

geopar={}; matpar={}; fabpar={};

(* Ref load is 1, whence external force is \(f=\lambda \times 1=\lambda\) *)

rfload={1};

(* Define incremental sol parameters - see above re cscale *)
nmax=100; \(\lambda\)max=2.; umax=2.; ell=0.25; adapt=False; acctol=0;
incpar={nmax, \(\lambda\)max, umax, ell, adapt, ell, ell, acctol, cscale};

(* Correction parameters are not used - set to empty list *)
corpar={};

(* Specify numerical (floating-point) computation *)
numer=True; options={numer};

(* Specify ref configuration as \(\lambda=\mu=1/\sqrt{2}\) on circle *)

refctl={1./Sqrt[2]}; refsta={1./Sqrt[2]};

(* Carry out nonlinear solution, which returns in soltab *)

{soltab,status}=NonLinSolDriver[problem,method,geopar,matpar,
  fabpar,incpar,corpar,refctl,refsta,rfload,options];
If [status!=" "],Print[status];

(* Print solution table *)
doflab={"\(\mu\)"}; digits={{5,3}};
PrintSolTable[problem,method,soltab,doflab,digits];

(* Show selective response plots *)

aspect=1; imgsiz=350; xrange=yrange={};
PlotSolTable[problem,method,soltab,{"\(\mu\)"","\(\lambda\)"},doflab,
  xrange,yrange,aspect,imgsiz,"Black",False];

Figure 27.6. Script “template” to run the Circle Game (CG) benchmark problem, with only
the unit circle as equilibrium path.
ClearAll[nmax, λ, μ];

(* Set tags: benchmark-problem, problem-variant ("BB" here)  
advancing-integration-scheme, increment-control-strategy. 
Internally: problem={probid,prbvar} & method={int,ics}; 
N.B.: ics links w/ cscale, declared {} for default *)

problem={"CG","BB"}; method={"FE","LC"}; cscale={};

(* No definition of structural properties: needed. 
Set geometry, material and fabrication parameters to empty lists *)

geopar={}; matpar={}; fabpar={};

(* Ref load is 1, whence external force is f=λ*1=λ *)

rfload={1};

(* Define incremental sol parameters - see above re cscale *)
nmax=50; λmax=2; umax=2; ell=0.25; adapt=False; acctol=0;
incpar={nmax,λmax,umax,ell,adapt,ell,ell,acctol,cscale};

(* Correction parameters are not used - set to empty list *)
corpar={};

(* Specify numerical (floating-point) computation *)

numer=True; options={numer};

(* Specify ref config as λ=μ=0 on straight line path λ=μ *)

refctl={0}; refsta={0};

(* Carry out nonlinear solution, which returns in soltab *)

{soltab,status}=NonLinSolDriver[problem,method,geopar,matpar,
fabpar,incpar,corpar,refctl,refsta,rfload,options];
If [status=" ",Print[status]]; 

(* Print solution table *)

doflab={"μ"}; digits={{5,3}};
PrintSolTable[problem,method,soltab,doflab,digits];

(* Show selective response plots *)

aspect=1; imgsiz=350; xrange=yrange={};
PlotSolTable[problem,method,soltab,"μ","λ",doflab,
xrange,yrange,aspect,imgsiz,"Black",False];

Figure 27.7. Script “template” to run BB (bifurcation branch) variant of the Circle Game (CG) benchmark problem. This variant has the unit circle and a straight line that passes through the origin as equilibrium paths.
Homework Exercises for Chapter 27
Corrective Methods: Implementation

All Exercises refer to the GeNoBeC (Geometrically Nonlinear Benchmarking with Corrector) program downloadable from this Chapter index.

EXERCISE 27.1