```
(* CLEAR EVERYTHING *)
ClearAll["Global`*"];
Needs["PlotLegends`"];

(* ============================================== *)
(* GAUSS RULE INFO MODULE (TAKEN FROM QuadSOR.nb) *)
(* ============================================== *)

QuadGaussRuleInfo[{rule_, numer_}, point_] :=
  Module[{ξ, η, p1, p2, i, j, w1, w2, m, info = {{Null, Null}, 0}},
   If[Length[rule] == 2, {p1, p2} = rule, p1 = p2 = rule];
   If[p1 < 0, Return[QuadNonProductGaussRuleInfo[{-p1, numer}, point]]];
   If[Length[point] == 2, {i, j} = point, m = point;
    j = Floor[(m - 1) / p1] + 1; i = m - p1 * (j - 1)];
   {ξ, w1} = LineGaussRuleInfo[{p1, numer}, i];
   {η, w2} = LineGaussRuleInfo[{p2, numer}, j];
   info = {{ξ, η}, w1 * w2};
   If[numer, Return[N[info]], Return[Simplify[info]]];];

LineGaussRuleInfo[{rule_, numer_}, point_] :=
  Module[{g2 = {-1, 1} / Sqrt[3], w3 = {5 / 9, 8 / 9, 5 / 9}, g3 = {-Sqrt[3 / 5], 0, Sqrt[3 / 5]}, w4 =
     {(1 / 2) - Sqrt[5 / 6] / 6, (1 / 2) + Sqrt[5 / 6] / 6, (1 / 2) + Sqrt[5 / 6] / 6, (1 / 2) - Sqrt[5 / 6] / 6},
    g4 = {-Sqrt[(3 + 2 * Sqrt[6 / 5]) / 7], -Sqrt[(3 - 2 * Sqrt[6 / 5]) / 7],
      Sqrt[(3 - 2 * Sqrt[6 / 5]) / 7], Sqrt[(3 + 2 * Sqrt[6 / 5]) / 7]}, g5 = {-Sqrt[5 + 2 * Sqrt[10 / 7]],
       -Sqrt[5 - 2 * Sqrt[10 / 7]], 0, Sqrt[5 - 2 * Sqrt[10 / 7]], Sqrt[5 + 2 * Sqrt[10 / 7]]} / 3,
    w5 = {322 - 13 * Sqrt[70], 322 + 13 * Sqrt[70], 512, 322 + 13 * Sqrt[70], 322 - 13 * Sqrt[70]} / 900,
    i = point, p = rule, info = {Null, 0}}, If[p == 1, info = {0, 2}];
   If[p == 2, info = {g2[[i]], 1}];
   If[p == 3, info = {g3[[i]], w3[[i]]}];
   If[p == 4, info = {g4[[i]], w4[[i]]}];
   If[p == 5, info = {g5[[i]], w5[[i]]}];
   If[numer, Return[N[info]], Return[Simplify[info]]];];

(* SHAPE FUNCTION DERIVATIVES MODULE *)
Quad4HRRingShapeFunDer[ncoor_, qcoor_, Jcons_] := Module[
 {r1, r2, r3, r4, z1, z2, z3, z4, ξ, η, Nf, dNr, dNz, A0, A1, A2, Jdet},
 {{r1, z1}, {r2, z2}, {r3, z3}, {r4, z4}} = ncoor; {ξ, η} = qcoor;
 Nf = {(1 - ξ) * (1 - η), (1 + ξ) * (1 - η), (1 + ξ) * (1 + η), (1 - ξ) * (1 + η)} / 4;
 A0 = ((r3 - r1) * (z4 - z2) - (r4 - r2) * (z3 - z1)) / 2;
 A1 = ((r3 - r4) * (z1 - z2) - (r1 - r2) * (z3 - z4)) / 2;
 A2 = ((r2 - r3) * (z1 - z4) - (r1 - r4) * (z2 - z3)) / 2;
 Jdet = (A0 + A1 * ξ + A2 * η) / 4; If [Jcons, Jdet = A0 / 4];
 dNr = {z2 - z4 + (z4 - z3) * ξ + (z3 - z2) * η, z3 - z1 + (z3 - z4) * ξ + (z1 - z4) * η,
  z4 - z2 + (z1 - z2) * ξ + (z4 - z1) * η, z1 - z3 + (z2 - z1) * ξ + (z2 - z3) * η} / (8 * Jdet);
 dNz = {r4 - r2 + (r3 - r4) * ξ + (r2 - r3) * η, r1 - r3 + (r4 - r3) * ξ + (r4 - r1) * η,
  r2 - r4 + (r2 - r1) * ξ + (r1 - r4) * η, r3 - r1 + (r1 - r2) * ξ + (r3 - r2) * η} / (8 * Jdet);
 Return[{Nf, dNr, dNz, Jdet}]
];

Quad4IsoPRingShapeFunDer[ncoor_, qcoor_, Jcons_] :=
  Module[{r1, r2, r3, r4, z1, z2, z3, z4, ξ, η, Nf, dNr, dNz, A0, A1, A2, Jdet},
   {{r1, z1}, {r2, z2}, {r3, z3}, {r4, z4}} = ncoor; {ξ, η} = qcoor;
   Nf = {(1 - ξ) * (1 - η), (1 + ξ) * (1 - η), (1 + ξ) * (1 + η), (1 - ξ) * (1 + η)} / 4;
   A0 = ((r3 - r1) * (z4 - z2) - (r4 - r2) * (z3 - z1)) / 2;
   A1 = ((r3 - r4) * (z1 - z2) - (r1 - r2) * (z3 - z4)) / 2;
   A2 = ((r2 - r3) * (z1 - z4) - (r1 - r4) * (z2 - z3)) / 2;
   Jdet = (A0 + A1 * ξ + A2 * η) / 4; If[Jcons, Jdet = A0 / 4];
   dNr = {z2 - z4 + (z4 - z3) * ξ + (z3 - z2) * η, z3 - z1 + (z3 - z4) * ξ + (z1 - z4) * η,
      z4 - z2 + (z1 - z2) * ξ + (z4 - z1) * η, z1 - z3 + (z2 - z1) * ξ + (z2 - z3) * η} / (8 * Jdet);
   dNz = {r4 - r2 + (r3 - r4) * ξ + (r2 - r3) * η, r1 - r3 + (r4 - r3) * ξ + (r4 - r1) * η,
      r2 - r4 + (r2 - r1) * ξ + (r1 - r4) * η, r3 - r1 + (r1 - r2) * ξ + (r3 - r2) * η} / (8 * Jdet);
   Return[{Nf, dNr, dNz, Jdet}]];

(* ============================================================ *)
(* ELEMENTAL STIFFNESS MATRIX MODULE (MODIFIED QuadSOR.nb CODE *)
```

```
(* ============================================================ *)

Quad4HRRingStiffness[ncoor_, Emat_, options_] := Module[
 {p = 2, numer = False, Jcons = False, Kfac = 1, qcoor, k, rk, zk, w, c,
  r1, r2, r3, r4, z1, z2, z3, z4, r0, z0, rBar, zBar, Nf, N1, N2, N3, N4,
  A0, Jdet, Be, Cmat, dNr1, dNr2, dNr3, dNr4, dNz1, dNz2,
  dNz3, dNz4, Ke, Ge, Fe, Finv, Ke0 = Table[0, {8}, {8}],
  modname = "Quad4HRRingStiffness: "},

 r0 = Mean[{r1, r2, r3, r4}];
 z0 = Mean[{z1, z2, z3, z4}];
 Cmat = Inverse[Emat];

 If [Length[options] == 1, {numer} = options];
 If [Length[options] == 2, {numer, p} = options];
 If [Length[options] == 3, {numer, p, Jcons} = options];
 If [Length[options] == 4, {numer, p, Jcons, Kfac} = options];
 If [p < 1 || p > 5, Print[modname, "illegal p:", p]; Return[Ke0]];
 {{r1, z1}, {r2, z2}, {r3, z3}, {r4, z4}} = ncoor;
 A0 = ((r3 - r1) * (z4 - z2) - (r4 - r2) * (z3 - z1)) / 2;
 If [numer && (A0 <= 0), Print[modname, "Neg or zero area"];
   Return[Ke0]]; Ke = Ke0;

 S = Table[0, {4}, {7}];
 Ge = Table[0, {7}, {8}];
 Fe = Table[0, {7}, {7}];

 For [k = 1, k <= p * p, k++,
   {qcoor, w} = QuadGaussRuleInfo[{p, numer}, k];
  {Nf, {dNr1, dNr2, dNr3, dNr4}, {dNz1, dNz2, dNz3, dNz4},
   Jdet} = Quad4HRRingShapeFunDer[ncoor, qcoor, Jcons];
   If [numer && (Jdet <= 0), Print[modname, "Neg or zero",
     " Gauss point Jacobian at k=", k]; Return[Ke0]];
   {N1, N2, N3, N4} = Nf;

   rk = r1 * N1 + r2 * N2 + r3 * N3 + r4 * N4;
   zk = z1 * N1 + z2 * N2 + z3 * N3 + z4 * N4;

   rBar = rk - r0;
   zBar = zk - z0;

   S = {{1, 0, 0, rBar, zBar, 0, 0},
     {0, 1, -zBar / rk, 0, 0, rBar, -(rk + rBar) * zBar / rk},
     {1, 0, 0, rk + rBar, zBar, 0, 0},
     {0, 0, 1, 0, 0, 0, rBar}};

   Be = {{ dNr1,  0, dNr2,  0, dNr3,  0, dNr4,  0},
     {  0, dNz1,  0, dNz2,  0, dNz3,  0, dNz4},
     {N1 / rk,  0, N2 / rk,  0, N3 / rk,  0, N4 / rk,  0},
     { dNz1, dNr1, dNz2, dNr2, dNz3, dNr3, dNz4, dNr4}};

   c = Kfac * w * rk * Jdet; If [numer, Be = N[Be]; c = N[c]];

   Ge += c * Transpose[S].Be;
   Fe += c * Transpose[S].(Cmat.S);

   ];

   Ke = Transpose[Ge].(Inverse[Fe].Ge);
   ClearAll[Ke0, Be]; Return[Ke] ];

Quad4IsoPRingStiffness[ncoor_, Emat_, options_] :=
  Module[{p = 2, numer = False, Jcons = False, Kfac = 1, qcoor, k, rk, w, c, r1, r2, r3, r4,
    z1, z2, z3, z4, Nf, N1, N2, N3, N4, A0, Jdet, Be, dNr1, dNr2, dNr3, dNr4, dNz1, dNz2,
```

```
    dNz3, dNz4, Ke, Ke0 = Table[0, {8}, {8}], modname = "Quad4IsoPRingStiffness: "},
   If[Length[options] == 1, {numer} = options];
   If[Length[options] == 2, {numer, p} = options];
   If[Length[options] == 3, {numer, p, Jcons} = options];
   If[Length[options] == 4, {numer, p, Jcons, Kfac} = options];
   If[p < 1 || p > 5, Print[modname, "illegal p:", p]; Return[Ke0]];
   {{r1, z1}, {r2, z2}, {r3, z3}, {r4, z4}} = ncoor;
   A0 = ((r3 - r1) * (z4 - z2) - (r4 - r2) * (z3 - z1)) / 2;
   If[numer && (A0 ≤ 0), Print[modname, "Neg or zero area"];
    Return[Ke0]]; Ke = Ke0;
   For[k = 1, k ≤ p * p, k++, {qcoor, w} = QuadGaussRuleInfo[{p, numer}, k];
    {Nf, {dNr1, dNr2, dNr3, dNr4}, {dNz1, dNz2, dNz3, dNz4}, Jdet} =
     Quad4IsoPRingShapeFunDer[ncoor, qcoor, Jcons];
    If[numer && (Jdet ≤ 0), Print[modname, "Neg or zero",
      " Gauss point Jacobian at k=", k]; Return[Ke0]];
    {N1, N2, N3, N4} = Nf; rk = r1 * N1 + r2 * N2 + r3 * N3 + r4 * N4;
    Be = {{dNr1, 0, dNr2, 0, dNr3, 0, dNr4, 0}, {0, dNz1, 0, dNz2, 0, dNz3, 0, dNz4}, {N1 / rk, 0,
       N2 / rk, 0, N3 / rk, 0, N4 / rk, 0}, {dNz1, dNr1, dNz2, dNr2, dNz3, dNr3, dNz4, dNr4}};
    c = Kfac * w * rk * Jdet; If[numer, Be = N[Be]; c = N[c]];
    Ke += c * Transpose[Be].(Emat.Be);]; ClearAll[Ke0, Be]; Return[Ke]];

Quad4IsoPSRIRingStiffness[ncoor_, Emat_, options_] :=
  Module[{p = 2, pSRI = p - 1, numer = False, Jcons = False, Kfac = 1, qcoor, k, rk, w,
    c, r1, r2, r3, r4, z1, z2, z3, z4, Nf, N1, N2, N3, N4, A0, Jdet, Be, dNr1, dNr2,
    dNr3, dNr4, dNz1, dNz2, dNz3, dNz4, Ev, Ed, Ke, KeV, KeD, Ke0 = Table[0, {8}, {8}],
    modname = "Quad4IsoPRingStiffness: "}, If[Length[options] == 1, {numer} = options];
   If[Length[options] == 2, {numer, p} = options];
   If[Length[options] == 3, {numer, p, Jcons} = options];
   If[Length[options] == 4, {numer, p, Jcons, Kfac} = options];
   If[p < 1 || p > 5, Print[modname, "illegal p:", p]; Return[Ke0]];
   {{r1, z1}, {r2, z2}, {r3, z3}, {r4, z4}} = ncoor;
   A0 = ((r3 - r1) * (z4 - z2) - (r4 - r2) * (z3 - z1)) / 2;
   If[numer && (A0 ≤ 0), Print[modname, "Neg or zero area"];
    Return[Ke0]]; Ke = Ke0; KeV = Ke0; KeD = Ke0;

   (* Volumetric part KeV *)
   Ev = Emat[[1]];
   For[k = 1, k ≤ p * p, k++, {qcoor, w} = QuadGaussRuleInfo[{p, numer}, k];
    {Nf, {dNr1, dNr2, dNr3, dNr4}, {dNz1, dNz2, dNz3, dNz4}, Jdet} =
     Quad4IsoPRingShapeFunDer[ncoor, qcoor, Jcons];
    If[numer && (Jdet ≤ 0), Print[modname, "Neg or zero",
      " Gauss point Jacobian at k=", k]; Return[Ke0]];
    {N1, N2, N3, N4} = Nf; rk = r1 * N1 + r2 * N2 + r3 * N3 + r4 * N4;
    Be = {{dNr1, 0, dNr2, 0, dNr3, 0, dNr4, 0}, {0, dNz1, 0, dNz2, 0, dNz3, 0, dNz4}, {N1 / rk, 0,
       N2 / rk, 0, N3 / rk, 0, N4 / rk, 0}, {dNz1, dNr1, dNz2, dNr2, dNz3, dNr3, dNz4, dNr4}};
    c = Kfac * w * rk * Jdet; If[numer, Be = N[Be]; c = N[c]];
    KeV += c * Transpose[Be].(Ev.Be);];

   (* Deviatoric part KeD *)
   Ed = Emat[[2]];
   For[k = 1, k ≤ pSRI * pSRI, k++, {qcoor, w} = QuadGaussRuleInfo[{pSRI, numer}, k];
    {Nf, {dNr1, dNr2, dNr3, dNr4}, {dNz1, dNz2, dNz3, dNz4}, Jdet} =
     Quad4IsoPRingShapeFunDer[ncoor, qcoor, Jcons];
    If[numer && (Jdet ≤ 0), Print[modname, "Neg or zero",
      " Gauss point Jacobian at k=", k]; Return[Ke0]];
    {N1, N2, N3, N4} = Nf; rk = r1 * N1 + r2 * N2 + r3 * N3 + r4 * N4;
    Be = {{dNr1, 0, dNr2, 0, dNr3, 0, dNr4, 0}, {0, dNz1, 0, dNz2, 0, dNz3, 0, dNz4}, {N1 / rk, 0,
       N2 / rk, 0, N3 / rk, 0, N4 / rk, 0}, {dNz1, dNr1, dNz2, dNr2, dNz3, dNr3, dNz4, dNr4}};
    c = Kfac * w * rk * Jdet; If[numer, Be = N[Be]; c = N[c]];
    KeD += c * Transpose[Be].(Ed.Be);];

   Ke = KeV + KeD;
   ClearAll[Ke0, Be]; Return[Ke]];
```

```
(* ========================================================================== *)
(* SOLVE THE AXISYMMETRIC RING ELEMENT PROBLEMS WITH DIFFERENT CONFIGURATIONS *)
(* ========================================================================== *)

RingSolver[Emat_, p_, P_, a_, b_, d_, Ner_,
   Nez_, etype_, nodeForces_, knownDispIndsInput_] := Module[{
    nnod, nele, nnodr, nnodz, dr, dz, k, ncoorAll, rcoorAll,
    zcoorAll, i, j, connect, q, ID, options,
    numer, Jcons, nc1, nc2, nc3, nc4, ncoor, Ke, map, globNod,
    Kg, rNum, cNum, knownDispInds, lenKnownDisp,
    nodeForcesReduced, KgReduced, oneVec, dropInd, uReduced,
    insInd, ur, uz, u, urPlotList, uzPlotList},

   (* total number of nodes and elements *)
   nnod = (Ner + 1) * (Nez + 1); nele = Ner * Nez;
   nnodr = Ner + 1; nnodz = Nez + 1;

   (* loop to define nodal coordinates *)
   dr = (b - a) / Ner;
   dz = d / Nez;
   k = 1;
   ncoorAll = Table[0, {nnod}, {1}];
   rcoorAll = Table[0, {nnod}, {1}];
   zcoorAll = Table[{0, 0}, {nnod}, {1}];
   For[i = 1, i ≤ nnodr, i++,
    For[j = 1, j ≤ nnodz, j++,
      rcoorAll[[k]] = a + (i - 1) * dr;
      zcoorAll[[k]] = (j - 1) * dz;
      ncoorAll[[k]] = {rcoorAll[[k]], zcoorAll[[k]]};
      k++;
     ];
   ];

   (* define the connectivity and
    an ID matrix to detrmine row and columnn location in Kg *)
   connect = Table[{0, 0, 0, 0}, {nele}, {1}];
   q = 0;
   For[k = 1, k ≤ nele, k++,
    connect[[k]] = {q + k, q + k + Nez + 1, q + k + Nez + 2, q + k + 1};
    If[IntegerQ[k / Nez], q++, Continue];
   ];

   ID = Table[{0, 0}, {nnod}, {1}];
   For[k = 1, k ≤ nnod, k++,
    ID[[k]] = {2 * k - 1, 2 * k};
   ];

   (* options set to be used in calling elemental stiffness module *)
   options = {numer = True, p, Jcons = False, 1};

   (* compute the elemental stiffness matrix *)
   Ke = Table[Table[0, {2 * 4}, {2 * 4}], {nele}, {1}];
   For[k = 1, k ≤ nele, k++,
    (* nodal coordinates *)
    nc1 = ncoorAll[[connect[[k]][[1]]]];
    nc2 = ncoorAll[[connect[[k]][[2]]]];
    nc3 = ncoorAll[[connect[[k]][[3]]]];
    nc4 = ncoorAll[[connect[[k]][[4]]]];
    ncoor = {nc1, nc2, nc3, nc4};
    (*Print["Nodal coordinates for element ",k," = ",ncoor//MatrixForm];*)
    (* compute the elemental stiffness matrix *)
    If[etype == "HR", Ke[[k]] = Quad4HRRingStiffness[ncoor, Emat, options]];
    If[etype == "ISOP", Ke[[k]] = Quad4IsoPRingStiffness[ncoor, Emat, options]];
    If[etype == "ISOPSRI", Ke[[k]] = Quad4IsoPSRIRingStiffness[ncoor, Emat, options]];
```

```
];

(* determine where Ke entries fit into Kg *)
map = Table[0, {nele}, {2 * 4}];
For[i = 1, i ≤ nele, i++,
 For[j = 1, j ≤ 4, j++,
   globNod = connect[[i, j]];
   map[[i, 2 * j - 1]] = ID[[globNod, 1]];
   map[[i, 2 * j]] = ID[[globNod, 2]];
  ];
];

(* assemble global stiffness matrix *)
Kg = Table[0, {2 * nnod}, {2 * nnod}];
For[k = 1, k ≤ nele, k++,  (* loop over elements *)
 For[i = 1, i ≤ 8, i++,
   For[j = 1, j ≤ 8, j++,

     (* row number *)
     rNum = map[[k, i]];
     (* column number *)
     cNum = map[[k, j]];

     (* assembly step *)
     Kg[[rNum, cNum]] += Ke[[k]][[i, j]];
    ];
  ];
];

(* ---------------------------------------------------------------------- *)
(* REDUCE THE STIFFNESS MATRIX FOR THE BOUNDARY CONDITIONS GIVEN BELOW *)
(* ---------------------------------------------------------------------- *)

(* now remove the appropriate rows and columns from Kg and nodeForces *)
KgReduced = Kg;
nodeForcesReduced = nodeForces;
knownDispInds = knownDispIndsInput;
lenKnownDisp = Dimensions[knownDispInds][[1]];
oneVec = Table[1, {lenKnownDisp}];
For[k = 1, k ≤ lenKnownDisp, k++,
 dropInd = knownDispInds[[k]];
 KgReduced = Drop[KgReduced, {dropInd, dropInd}, {dropInd, dropInd}];
 nodeForcesReduced = Drop[nodeForcesReduced, {dropInd, dropInd}];
 knownDispInds = knownDispInds - oneVec;
];

(* SOLVE THE SYSTEM OF EQUATIONS *)
uReduced = LinearSolve[KgReduced, nodeForcesReduced];
(* Print["Reduced Displacement Vector = ",uReduced//MatrixForm]; *)

(* Insert zeroed displacements back into displacement vector for plotting *)
u = uReduced;
For[k = lenKnownDisp, k ≥ 1, k--,
 knownDispInds = knownDispInds + oneVec;
 insInd = knownDispInds[[k]];
 u = Insert[u, 0, insInd];
];
(* Print["Displacements Including BC data = ",u//MatrixForm]; *)

(* separate u into ur and uz *)
ur = Table[0, {nnod}];
uz = Table[0, {nnod}];
For[k = 1, k ≤ nnod, k++,
 ur[[k]] = u[[2 * k - 1]];
```

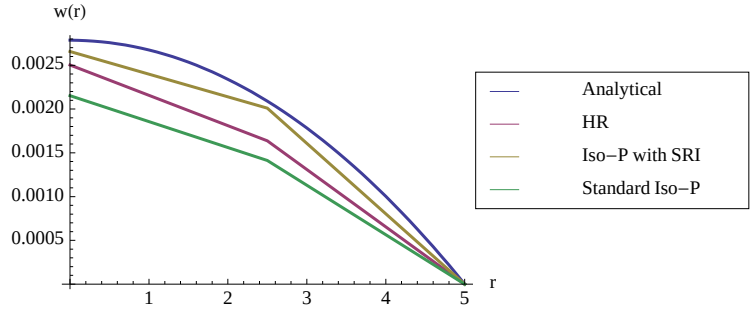
```
    uz[[k]] = u[[2 * k]];
   ];

   (* PLOT RESULTS ALONG THE
    MIDPLANE:  ***** Nez MUST BE AN EVEN NUMBER FOR THIS TO WORK! ***** *)

   (* must define plot lists contining midpoint
    displacement values and nodal coordinates (only vs r is necessary) *)
   urPlotList = Table[0, {nnodr}, {2}];
   uzPlotList = Table[0, {nnodr}, {2}];
   q = 1;
   For[k = (Nez / 2 + 1), k ≤ nnod, k += nnodz,
    urPlotList[[q, 1]] = rcoorAll[[k]];
    urPlotList[[q, 2]] = ur[[k]];
    uzPlotList[[q, 1]] = rcoorAll[[k]];
    uzPlotList[[q, 2]] = uz[[k]];
    q++;
   ];

   (* return some values for plotting and further analysis *)
   returnValues = Table[0, {5}];
   returnValues = {rcoorAll, ur, uz, urPlotList, uzPlotList};
   Return[returnValues];

  ];
(* =============== *)
(* PLOTTING MODULE *)
(* =============== *)

multiPlot[w_, uzPlotListHR_, uzPlotListSRI_, uzPlotListISOP_] :=
 Module[{step, wList, rList, wPlotList, compPlot},
  step = 0.1;
  wList = Table[N[w], {r, a, b, step}];
  rList = Table[r, {r, a, b, step}];
  wPlotList = Table[0, {2}, {1 / step}];
  wPlotList[[1]] = rList;
  wPlotList[[2]] = wList;
  wPlotList = Transpose[wPlotList];

  compPlot = ListLinePlot[{wPlotList, uzPlotListHR, uzPlotListSRI, uzPlotListISOP},
    PlotStyle → Thickness[0.0075],
    PlotLegend -> {"Analytical", "HR", "Iso-P with SRI", "Standard Iso-P"},
    LegendShadow → None,
    LegendSize → {1, .5},
    LegendPosition → {0.7, -.25},
    PlotLabel → "Element Comparison:  Midplane z displacement",
    AxesLabel → {"r", "w(r)"}
   ];
  Return[compPlot];
 ]


(* ================================================ *)
(* SIMPLY SUPPRTED PLATE SUBJECTED TO BENDING MOMENT *)
(* ================================================ *)

(* define material properties *)
Ym = 70 * 10^9;  (* Young's modulus for Aluminum *)
nu = 0.35;   (* Poisson's ration for Aluminum *)
(*Ym=1000;
nu=(1/3);*)

(* define the material matrix for plane stress *)
Emat = (Ym / ((1 + nu) * (1 - 2 * nu))) *
```

```
   {{1 - nu, nu, nu, 0}, {nu, 1 - nu, nu, 0}, {nu, nu, 1 - nu, 0}, {0, 0, 0, 0.5 * (1 - 2 * nu)}};

(* number of Gauss points *)
p = 2;

(* plate inner radius a, outer radius b, and thickness d *)
a = 0; b = 5; d = 1;

(* number of elements in the r and z direction *)
Ner = 2; Nez = 2;

(* total number of nodes and elements *)
nnod = (Ner + 1) * (Nez + 1); nele = Ner * Nez;
nnodr = Ner + 1; nnodz = Nez + 1;

(* Force *)
P = 10 000 000;

(* force vector *)
nodeForces = Table[0, {2 * nnod}];
nodeForces[[2 * nnod - 1]] = P;
nodeForces[[2 * nnod - 2 * nnodz + 1]] = -P;

(* displacement vector will be of the form {{u1x},{u1y},{u2x},{u2y},.....} *)
(* The center nodes will have displacements constrained in the r direction.  The
 outer node in the midplane will have displacement constrained in the z direction
 for the simply supported case (this requires an even number of elements).  These
 boundary conditions allow us to cross out rows and columns in the stiffness
 matrix to reduce the size of the linear equation system.  ID numbers of the
 nodal quantities that are zeroed must be recorded for plotting purposes later,
and to track which nodes the values in the returned displacement vector correspond to. *)

(* indeces of displacements to be zeroed:  *)
knownDispInds = Table[0, {nnodz + 2}];
(* center nodes *)
q = 1;
For[k = 1, k ≤ (2 * nnodz), k = k + 2,
  knownDispInds[[q]] = k;
  q++;
 ];
(* outer center node *)
knownDispInds[[q]] = 2 * nnod - nnodz; q++;
knownDispInds[[q]] = knownDispInds[[q - 1]] + 1;

(* RESULT USING HR WITH 7 STRESS PARAMETERS *)
{rcoorAllHR, urHR, uzHR, urPlotListHR, uzPlotListHR} =
  RingSolver[Emat, p, P, a, b, d, Ner, Nez, "HR", nodeForces, knownDispInds];

(* SRI ISO-P RESULT *)
(* need to define volumetric and deviatoric material matrices *)
Ed = Ym * nu / ((1 + nu) * (1 - 2 * nu)) * {{1, 1, 1, 0}, {1, 1, 1, 0}, {1, 1, 1, 0}, {0, 0, 0, 0}};
Ev = Ym / (2 * (1 + nu)) * {{2, 0, 0, 0}, {0, 2, 0, 0}, {0, 0, 2, 0}, {0, 0, 0, 1}};
EmatSRI = {Ed, Ev};
{rcoorAllSRI, urSRI, uzSRI, urPlotListSRI, uzPlotListSRI} =
  RingSolver[EmatSRI, p, P, a, b, d, Ner, Nez, "ISOPSRI", nodeForces, knownDispInds];

(* STANDARD ISO-P RESULT *)
{rcoorAllISOP, urISOP, uzISOP, urPlotListISOP, uzPlotListISOP} =
  RingSolver[Emat, p, P, a, b, d, Ner, Nez, "ISOP", nodeForces, knownDispInds];

(* ANALYTICAL SOLUTION:  Plate with no hole in center *)
Dc = (Ym * d^3) / (12 * (1 - nu^2));
M = P * d / b;
row = r / b;
```

```
C1 = 1 - row^2;
w = (M * b^2) / (2 * Dc * (1 + nu)) * C1;

compPlot = multiPlot[w, uzPlotListHR, uzPlotListSRI, uzPlotListISOP];
Print["===== Simply Supported Plate With Edge Moment ====="];
Show[compPlot]
```

```
===== Simply Supported Plate With Edge Moment =====
```

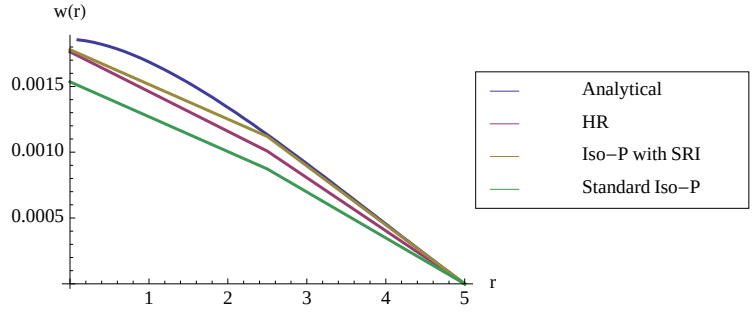
Element Comparison: Midplane z displacement



```
(* ===================================== *)
(* SIMPLY SUPPORTED PLATE WITH POINT LOAD *)
(* ===================================== *)

(* NOTE:  Many definitions are carried over from
  previous case (geometry, number of gauss points, Emat, etc).
   If different definitions are desired they must be stated below. *)

(* Force *)
Ppl = P / (2 * Pi); (* for point load case divide by 2*pi *)

(* force vector *)
nodeForces = Table[0, {2 * nnod}];
nodeForces[[2 * nnodz]] = Ppl;

(* RESULT USING HR, SRI, AND ISO-P *)
{rcoorAllHR, urHR, uzHR, urPlotListHR, uzPlotListHR} =
  RingSolver[Emat, p, Ppl, a, b, d, Ner, Nez, "HR", nodeForces, knownDispInds];
{rcoorAllSRI, urSRI, uzSRI, urPlotListSRI, uzPlotListSRI} =
  RingSolver[EmatSRI, p, Ppl, a, b, d, Ner, Nez, "ISOPSRI", nodeForces, knownDispInds];
{rcoorAllISOP, urISOP, uzISOP, urPlotListISOP, uzPlotListISOP} =
  RingSolver[Emat, p, Ppl, a, b, d, Ner, Nez, "ISOP", nodeForces, knownDispInds];

(* ANALYTICAL SOLUTION *)
Dc = (Ym * d^3) / (12 * (1 - nu^2));
row = r / b;
C2 = (row^2) * Log[row];
w = P * (b^2) / (16 * Pi * Dc) * ((3 + nu) / (1 + nu) * C1 + 2 * C2);

compPlot = multiPlot[w, uzPlotListHR, uzPlotListSRI, uzPlotListISOP];
Print["===== Simply Supported Plate With Point Load ====="];
Show[compPlot]
```

```
===== Simply Supported Plate With Point Load =====
```

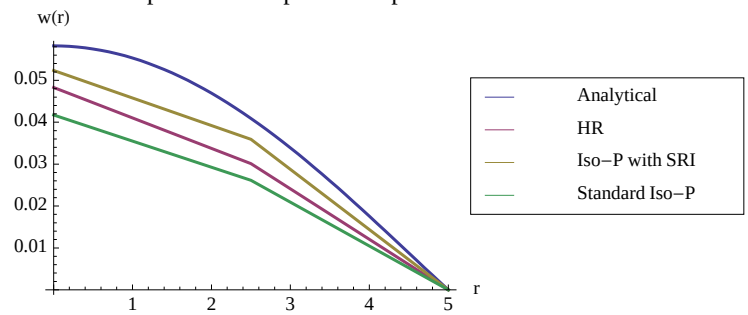
Element Comparison: Midplane z displacement



```
(* ========================================== *)
(* SIMPLY SUPPORTED PLATE WITH DISTRIBUTED LOAD *)
(* ========================================== *)

(* NOTE:  Many definitions are carried over from
  previous case (geometry, number of gauss points, Emat, etc).
   If different definitions are desired they must be stated below. *)


(* NODAL FORCES:
  The load P is a force per unit area distributed evenly over the surface of the disk.   *)
rWidth = b / (2 * Ner);
Arat = Table[0, {nnodr}];
Arat[[1]] = Pi * (rWidth^2) / (2 * Pi);
r1 = rWidth;
r2 = r1 + 2 * rWidth;
For[k = 2, k ≤ (nnodr - 1), k++,
  Arat[[k]] = Pi * (r2^2 - r1^2) / (2 * Pi);
  r1 += 2 * rWidth;
  r2 += 2 * rWidth;
 ];
r2 = r1 + rWidth;
Arat[[nnodr]] = Pi * (r2^2 - r1^2) / (2 * Pi);

nodeForces = Table[0, {2 * nnod}];
For[k = 1, k ≤ nnodr, k++,
  nodeForces[[2 * nnodz * k]] = P * Arat[[k]];
 ];


(* RESULT USING HR, SRI, AND ISO-P *)
{rcoorAllHR, urHR, uzHR, urPlotListHR, uzPlotListHR} =
  RingSolver[Emat, p, Ppl, a, b, d, Ner, Nez, "HR", nodeForces, knownDispInds];
{rcoorAllSRI, urSRI, uzSRI, urPlotListSRI, uzPlotListSRI} =
  RingSolver[EmatSRI, p, Ppl, a, b, d, Ner, Nez, "ISOPSRI", nodeForces, knownDispInds];
{rcoorAllISOP, urISOP, uzISOP, urPlotListISOP, uzPlotListISOP} =
  RingSolver[Emat, p, Ppl, a, b, d, Ner, Nez, "ISOP", nodeForces, knownDispInds];

(* ANALYTICAL SOLUTION *)
Dc = (Ym * d^3) / (12 * (1 - nu^2));
row = r / b;
C0 = 1 - row^4;
w = P * (b^4) / (64 * Dc * (1 + nu)) * (2 * (3 + nu) * C1 - (1 + nu) * C0);

compPlot = multiPlot[w, uzPlotListHR, uzPlotListSRI, uzPlotListISOP];
Print["===== Simply Supported Plate With Distributed Load ====="];
Show[compPlot]
```

```
===== Simply Supported Plate With Distributed Load =====
```

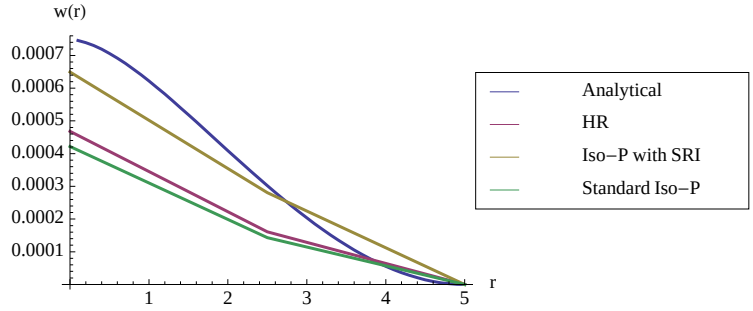
Element Comparison: Midplane z displacement



```
(* ============================== *)
(* CLAMPED PLATE WITH POINT LOAD *)
(* ============================== *)

(* NOTE:  Many definitions are carried over from
  previous case (geometry, number of gauss points, Emat, etc).
   If different definitions are desired they must be stated below. *)

(* Force *)
Ppl = P / (2 * Pi); (* for point load case divide by 2*pi *)

(* force vector *)
nodeForces = Table[0, {2 * nnod}];
nodeForces[[2 * nnodz]] = Ppl;

(* indeces of node & direction with zero displacment boundary conditions *)
knownDispInds = Table[0, {3 * nnodz}];
For[k = 1, k ≤ 2 * nnodz, k++,
  If[Element[k / 2, Integers], Continue, knownDispInds[[(k + 1) / 2]] = k];
  knownDispInds[[k + nnodz]] = k + 2 * (nnod - nnodz);
 ];

(* RESULT USING HR, SRI, AND ISO-P *)
{rcoorAllHR, urHR, uzHR, urPlotListHR, uzPlotListHR} =
  RingSolver[Emat, p, Ppl, a, b, d, Ner, Nez, "HR", nodeForces, knownDispInds];
{rcoorAllSRI, urSRI, uzSRI, urPlotListSRI, uzPlotListSRI} =
  RingSolver[EmatSRI, p, Ppl, a, b, d, Ner, Nez, "ISOPSRI", nodeForces, knownDispInds];
{rcoorAllISOP, urISOP, uzISOP, urPlotListISOP, uzPlotListISOP} =
  RingSolver[Emat, p, Ppl, a, b, d, Ner, Nez, "ISOP", nodeForces, knownDispInds];

(* ANALYTICAL SOLUTION *)
w = P * (b^2) / (16 * Pi * Dc) * (1 - row^2 + 2 * (row^2) * Log[row]);

compPlot = multiPlot[w, uzPlotListHR, uzPlotListSRI, uzPlotListISOP];
Print["===== Clamped Plate With Point Load ====="];
Show[compPlot]
```

```
===== Clamped Plate With Point Load =====
```

Element Comparison: Midplane z displacement



```
(* ================================== *)
(* CLAMPED PLATE WITH DISTRIBUTED LOAD *)
(* ================================== *)

(* NOTE:  Many definitions are carried over from
  previous case (geometry, number of gauss points, Emat, etc).
   If different definitions are desired they must be stated below. *)


(* NODAL FORCES:
  The load P is a force per unit area distributed evenly over the surface of the disk.   *)
rWidth = b / (2 * Ner);
Arat = Table[0, {nnodr}];
Arat[[1]] = Pi * (rWidth^2) / (2 * Pi);
r1 = rWidth;
r2 = r1 + 2 * rWidth;
For[k = 2, k ≤ (nnodr - 1), k++,
  Arat[[k]] = Pi * (r2^2 - r1^2) / (2 * Pi);
  r1 += 2 * rWidth;
  r2 += 2 * rWidth;
 ];
r2 = r1 + rWidth;
Arat[[nnodr]] = Pi * (r2^2 - r1^2) / (2 * Pi);

nodeForces = Table[0, {2 * nnod}];
For[k = 1, k ≤ nnodr, k++,
  nodeForces[[2 * nnodz * k]] = P * Arat[[k]];
 ];


(* RESULT USING HR, SRI, AND ISO-P *)
{rcoorAllHR, urHR, uzHR, urPlotListHR, uzPlotListHR} =
  RingSolver[Emat, p, Ppl, a, b, d, Ner, Nez, "HR", nodeForces, knownDispInds];
{rcoorAllSRI, urSRI, uzSRI, urPlotListSRI, uzPlotListSRI} =
  RingSolver[EmatSRI, p, Ppl, a, b, d, Ner, Nez, "ISOPSRI", nodeForces, knownDispInds];
{rcoorAllISOP, urISOP, uzISOP, urPlotListISOP, uzPlotListISOP} =
  RingSolver[Emat, p, Ppl, a, b, d, Ner, Nez, "ISOP", nodeForces, knownDispInds];

(* ANALYTICAL SOLUTION *)
(*p0=P/(Pi*b^2);*)
w = P * (b^4) / (64 * Dc) * (1 - row^2)^2;

compPlot = multiPlot[w, uzPlotListHR, uzPlotListSRI, uzPlotListISOP];
Print["===== Clamped Plate With Distributed Load ====="];
Show[compPlot]
```

```
===== Clamped Plate With Distributed Load =====
```

Element Comparison: Midplane z displacement