```
Manipulate[
 (*updated may 30, 2014*)
 gtick; (*tracked symbol*)

 {finalDisplayImage, u, state, forceGrid, grid, normf, cpuTimeUsed, stepNumber, mask, Minv,
   splittingNmatrix, factorLmatrix, factorDmatrix, factorUmatrix, iterationMatrix,
   systemAmatrix, rightHandVector, steepestDescentR, conjugateGradientR,
   conjugateGradientP, conjugateGradientZ, residualPlotData, relativeResidual, residual,
   preConditionerMatrix, residualPlotLimits, SORChebyomega, gstatusMessage, gtick} =
  processPDE[u, state, event, forceGrid, grid, normf, cpuTimeUsed, stepNumber,
   mask, rightHandVector, Minv, splittingNmatrix, factorLmatrix, factorDmatrix,
   factorUmatrix, iterationMatrix, systemAmatrix, steepestDescentR, conjugateGradientR,
   conjugateGradientP, conjugateGradientZ, Unevaluated@residualPlotData,
   SORomegaUserValue, SORChebyomega, residualPlotLimits, preConditionerMatrix,
   relativeResidual, residual, fillIn, nonStationarySolver, stationarySolver,
   solverType, splittingOrRelaxation, preconditioner, addFaceGrids,
   zAxisScale, plotPerformanceGoal, hx, hy, lenX, lenY, centerGrid, a, b, c,
   x0, y0, stdx, stdy, initialSolution, forceTermSelection, typeOfplotToShow,
   plotToShow, northBCtype, northbc, northBCconstantValue, westBCtype, westbc,
   westBCconstantValue, eastBCtype, eastbc, eastBCconstantValue, southBCtype,
   southbc, southBCconstantValue, toleranceConstant, finalDisplayImage, del, gtick];
 FinishDynamic[];

 Framed[finalDisplayImage ,
  FrameStyle -> Directive[Thickness[.005], Gray], ImageMargins → 0],

 Evaluate@With[{
    (*-----------------------------*)
    (*--- pde3 PLOT optioons ------*)
    (*-----------------------------*)
    plotOptionsMacro = Grid[{
       {
        Grid[{
          {
           Grid[{
              {Style["plot", 10],
               PopupMenu[Dynamic[plotToShow, {plotToShow = #; gtick += del} &],
                {"solution" → Style["solution 3D", 12],
                 "solution data" → Style["solution data", 12],
                 "system matrix information" → Style["system matrix", 12],
                 "residual" → Style["residual", 12],
                 "convergence" → Style["convergence", 12]
                }, ImageSize -> All, ContinuousAction -> False]
              },
              {
               Style["style", 10],
               PopupMenu[Dynamic[
                 typeOfplotToShow, {typeOfplotToShow = #; gtick += del} &],
                { "ListPlot3D1" → Style["ListPlot3D (1)", 12],
                  "ListPlot3D2" → Style["ListPlot3D (2)", 12],
                  "ListPlot3D3" → Style["ListPlot3D (3)", 12],
                  "ArrayPlot" → Style["ArrayPlot", 12],
```

```
            "ListContourPlot" → Style["ListContour (1)    ", 12],
            "ListContourPlot w/o labels" → Style["ListContour (2)", 12],
             "ListDensityPlot" → Style["ListDensity", 12]
           },
           ImageSize -> All, ContinuousAction -> False, Enabled →
            Dynamic[plotToShow == "solution" || plotToShow == "residual"]]
         }
        }, Alignment → Left, Spacings → {0.2, 0}
       ],

       Grid[{
         {RadioButtonBar[Dynamic[
            plotPerformanceGoal, {plotPerformanceGoal = #; gtick += del} &],
           {"Speed" -> Text@Style["speed", 11], "Quality" ->
             Text@Style["quality", 11]}, Appearance → "Vertical"]
         },
         {Style["render", 11]}
        }],

       Grid[{{Checkbox[Dynamic[addFaceGrids, {addFaceGrids = #; gtick += del} &]]
         },
         {Style[Column[{"face", "grids"}], 11]}
        }],

       Grid[{{Checkbox[Dynamic[zAxisScale, {zAxisScale = #; gtick += del} &]]
         },
         {Style[Column[{"zoom", ""}], 11]}
        }]
      }
     }, Alignment → Center, Spacings → {.7, .2},
     Frame → {All}, FrameStyle -> Directive[Thickness[.005], Gray]
    ]
   }
  }],
(*------------------------*)
(*--- TOP ROW macro -----*)
(*------------------------*)
topRowMacro = Item[Grid[{
    {
     Button[Text[Style["solve", 12]], {event = "run_button";
       If[Not[state == "PAUSE"], state = "INIT"]; gtick += del}, ImageSize -> {50, 35}],
     Button[Text[Style["pause", 12]], {event = "pause_button"; gtick += del},
      ImageSize -> {52, 35}],
     Button[Text[Style["step", 12]], {event = "step_button"; If[Not[state == "RUNNING"],
        state = "RUNNING"]; gtick += del}, ImageSize -> {48, 35}],
     Button[Text[Style["reset", 12]], {event = "reset"; gtick += del},
      ImageSize -> {48, 35}],
     SpanFromLeft
    }}, Alignment → Center, Spacings → {0.1, 0}
  ], Alignment → {Center, Top}
 ],
(*--------------------------*)
```

```
(*--- geometryMacro macro --*)
(*--------------------------*)
geometryMacro = Item[Grid[{
    {
     Grid[{
       {
        Grid[{
          {
           Row[{Text@Style[Row[{Style["h", Italic]Style["x",Italic] }], 12], Spacer[2],
             SetterBar[Dynamic[hx,
               {hx = #; SORomegaUserValue = setOptimalSORomega[hx, hy, lenX, lenY];
                 event = "reset"; gtick += del} &], {1 / 4, 1 / 8, 1 / 16}]}],

           Spacer[25],

           Row[{Text@Style[Row[{Style["x", Italic], " length" }], 12], Spacer[2],
             SetterBar[Dynamic[lenX,
               {lenX = #; SORomegaUserValue = setOptimalSORomega[hx, hy, lenX, lenY];
                 event = "reset"; gtick += del} &], {1, 2}]}]
          }
          ,
          {
           Row[{Text@Style[Row[{Style["h", Italic]Style["y",Italic]}], 12], Spacer[2],
             SetterBar[Dynamic[hy,
               {hy = #; SORomegaUserValue = setOptimalSORomega[hx, hy, lenX, lenY];
                 event = "reset"; gtick += del} &], {1 / 4, 1 / 8, 1 / 16}]}],

           Spacer[25],
           Row[{Text@Style[Row[{Style["y", Italic], " length" }], 12], Spacer[2],
             SetterBar[Dynamic[lenY,
               {lenY = #; SORomegaUserValue = setOptimalSORomega[hx, hy, lenX, lenY];
                 event = "reset"; gtick += del} &], {1, 2}]}]
          }
         }, Spacings → {.7, 0}
        ],
        Grid[{
          {Style[Column[{"center", "grid"}, Alignment → Center], 11]},
          {Checkbox[
            Dynamic[centerGrid, {centerGrid = #; event = "reset"; gtick += del} &]]}
         }, Spacings → {0, .4}], SpanFromLeft
       }
      }, Spacings → {.8, 0}, Alignment → Center], SpanFromLeft
    }
    ,
    {
     Grid[{
       {
        RadioButtonBar[Dynamic[northBCtype, {northBCtype = #; event = "reset";
             gtick += del} &], {"Dirichlet" → Text@Style["Dirichlet", 10],
          "Neumann" → Text@Style["Neumann", 10]}, Appearance → "Horizontal"
```

```
   ], SpanFromLeft
  },
  {
   PopupMenu[Dynamic[northbc, {northbc = #; event = "reset"; gtick += del} &],
    {
     (1.0) & → Style["a", Italic, 11],
     (#) & → Style["a x", Italic, 11],
     (#^2) & → Style[Row[{Style["a ", Italic], Style["x", Italic]^2}], 11],
     (Cos[Pi #]) & → Style[
       Row[{Style["a", Italic], " cos(π ", Style["x", Italic], ")"}], 11],
     (Cos[2 Pi #]) & → Style[Row[{Style["a", Italic], " cos(2 π ",
         Style["x", Italic], ")"}], 11]
    }, ImageSize -> All, ContinuousAction -> False
   ], SpanFromLeft
  },
  {
   Grid[{
     {
      Text@Style["a", Italic, 12],
      Manipulator[Dynamic[northBCconstantValue,
         {northBCconstantValue = #; event = "reset"; gtick += del} &],
        {-20, 20, 0.1}, ImageSize -> Tiny, ContinuousAction -> False],
      Text@Style[Dynamic@padIt1[northBCconstantValue, {3, 1}], 10],
      Button[Text@Style["zero", 11], {northBCconstantValue = 0.0;
         event = "reset"; gtick += del}, ImageSize → {45, 20}],
      Button[Text@Style["one", 11], {northBCconstantValue = 1.0;
         event = "reset"; gtick += del}, ImageSize → {45, 20}]
     }
    }, Spacings → {.1, 0}, Alignment → Center, Frame → None], SpanFromLeft
  }
 }, Frame → None, Spacings → {0.1, 0}, Alignment → Center
], SpanFromLeft
}
,
{
 Grid[{
   {
    RadioButtonBar[Dynamic[westBCtype, {westBCtype = #; event = "reset";
        gtick += del} &], {"Dirichlet" → Text@Style["Dirichlet", 10],
      "Neumann" → Text@Style["Neumann", 10]}, Appearance → "Horizontal"]
   },
   {
    PopupMenu[Dynamic[westbc, {westbc = #; event = "reset"; gtick += del} &],
     { (1.0) & → Style["a", Italic, 11],
      (#) & → Style["a y", Italic, 11],
      (#^2) & → Style[Row[{Style["a ", Italic], Style["y", Italic]^2}], 11],
      (Cos[Pi #]) & → Style[
        Row[{Style["a", Italic], " cos(π ", Style["y", Italic], ")"}], 11],
      (Cos[2 Pi #]) & → Style[Row[{Style["a", Italic], " cos(2 π ",
          Style["y", Italic], ")"}], 11]
```

```
      }, ImageSize -> All, ContinuousAction -> False
     ]
    },
    {Grid[{
       {
        Grid[{

           {Text@Style["a", Italic, 12],
            Manipulator[Dynamic[westBCconstantValue,
               {westBCconstantValue = #; event = "reset"; gtick += del} &],
             {-20, 20, 0.1}, ImageSize -> Tiny, ContinuousAction -> False
            ],
            Text@Style[Dynamic@padIt1[westBCconstantValue, {3, 1}], 10]
           },
           {
            Row[{
               Button[Text@Style["zero", 11], {westBCconstantValue = 0.0;
                 event = "reset"; gtick += del}, ImageSize → {45, 20}]
               , Spacer[2],
               Button[Text@Style["one", 11], {westBCconstantValue = 1.0;
                 event = "reset"; gtick += del}, ImageSize → {45, 20}]
              }], SpanFromLeft
           }
          }, Spacings → {.1, 0}, Alignment → Center]
       }
      }, Alignment → Center, Spacings → {0, 0}
     ]
    }
   }
  ]
  ,
  Grid[{
     {
      RadioButtonBar[Dynamic[eastBCtype, {eastBCtype = #; event = "reset";
           gtick += del} &], {"Dirichlet" → Text@Style["Dirichlet", 10], "Neumann" →
          Text@Style["Neumann", 10]}, Appearance → "Horizontal"], SpanFromLeft
    },
    {
     PopupMenu[Dynamic[eastbc, {eastbc = #; event = "reset"; gtick += del} &],
       { (1.0) & → Style["a", Italic, 11],
        (#) & → Style["a y", Italic, 11],
        (#^2) & → Style[Row[{Style["a ", Italic], Style["y", Italic]^2}], 11],
        (Cos[Pi #]) & → Style[
          Row[{Style["a", Italic], " cos(π ", Style["y", Italic], ")"}], 11],
        (Cos[2 Pi #]) & → Style[Row[{Style["a", Italic], " cos(2 π ",
             Style["y", Italic], ")"}], 11]
      }, ImageSize -> All, ContinuousAction -> False
     ], SpanFromLeft
    },
    {Grid[{
       {
```

```
          Grid[{
            {
             Text@Style["a", Italic, 12],
             Manipulator[Dynamic[eastBCconstantValue,
                {eastBCconstantValue = #; event = "reset"; gtick += del} &],
              {-20, 20, 0.1}, ImageSize -> Tiny, ContinuousAction -> False],
             Text@Style[Dynamic@padIt1[eastBCconstantValue, {3, 1}], 10]
            },
            {
             Row[{Button[Text@Style["zero", 11], {eastBCconstantValue = 0.0;
                  event = "reset"; gtick += del}, ImageSize → {45, 20}],
                Button[Text@Style["one", 11], {eastBCconstantValue = 1.0;
                  event = "reset"; gtick += del}, ImageSize → {45, 20}]
              }], SpanFromLeft
            }
           }, Spacings → {.2, 0}, Alignment → Center]
         }
        }, Alignment → Center, Spacings → {0, 0}
       ],
       SpanFromLeft
      }
     }]
   },
   {
    Grid[{
      {
       RadioButtonBar[Dynamic[southBCtype, {southBCtype = #; event = "reset";
            gtick += del} &], {"Dirichlet" → Text@Style["Dirichlet", 10],
          "Neumann" → Text@Style["Neumann", 10]}, Appearance → "Horizontal"]
      },
      {
       PopupMenu[Dynamic[southbc, {southbc = #; event = "reset"; gtick += del} &],
        { (1.0) & → Style["a", Italic, 11],
         (#) & → Style["a x", Italic, 11],
         (#^2) & → Style[Row[{Style["a ", Italic], Style["x", Italic]^2}], 11],
         (Cos[Pi #]) & → Style[
           Row[{Style["a", Italic], " cos(π ", Style["x", Italic], ")"}], 11],
         (Cos[2 Pi #]) & → Style[Row[{Style["a", Italic], " cos(2 π ",
             Style["x", Italic], ")"}], 11]
        }, ImageSize -> All, ContinuousAction -> False
       ]
      },
      {Grid[{
         {Text@Style["a", Italic, 12],
          Manipulator[Dynamic[southBCconstantValue,
             {southBCconstantValue = #; event = "reset"; gtick += del} &],
           {-20, 20, 0.1}, ImageSize -> Tiny, ContinuousAction -> False],
          Text@Style[Dynamic@padIt1[southBCconstantValue, {3, 1}], 10],
          Button[Text@Style["zero", 11], {southBCconstantValue = 0.0;
             event = "reset"; gtick += del}, ImageSize → {45, 20}],
          Button[Text@Style["one", 11], {southBCconstantValue = 1.0;
```

```
                event = "reset"; gtick += del}, ImageSize → {45, 20}]
             }
            }, Spacings → {.2, 0}
           ]
          }
         }
        ], SpanFromLeft
       }
      }, Spacings → {2.9, .6}, Alignment → Center, Frame → All,
      FrameStyle -> Directive[Thickness[.005], Gray]], Alignment → {Center, Top}
    ],
   (*-----------------------------------*)
   (*--- sourceMacro           macro ---*)
   (*-----------------------------------*)
   sourceMacro = Item[Grid[{
       {Item[PopupMenu[Dynamic[forceTermSelection,
            {forceTermSelection = #; event = "reset"; gtick += del} &],
          {1 → Style["a", Italic, 12],
           2 → Style["x y", Italic, 12],
           3 → Style[Row[{Style["a", Italic], " exp (",
                Row[{Style["x", Italic], " - ", (Style["x", Italic]"0")^2}] /
                 ("2 σ"Style["x",Italic])^2, " + ", Row[{Style["y", Italic], " - ",
                   (Style["y", Italic]"0")^2}] / ("2 σ")^2, ")"}], 12],
           4 → Style[Row[{Style["a", Italic], " (cos(", Style["b", Italic],
                " π ", Style["x", Italic], ") + sin(", Style["c", Italic],
                " π ", Style["y", Italic], "))"}], 12],
           5 → Style[Row[{Style["a", Italic], " cos(", Style["b", Italic],
                " π ", Style["x", Italic], ") * sin(",
                Style["c", Italic], " π ", Style["y", Italic], ")"}], 12]
          }, ImageSize → {ContentSizeW, ContentSizeH - 365}, ContinuousAction -> False],
         Alignment → {Center}
        ], SpanFromLeft
       },
       {
        Grid[{
          {
           Text@Style["a", Italic, 12],
           Manipulator[Dynamic[a, {a = #; event = "reset"; gtick += del} &],
            {-10., 10., 0.1}, ImageSize → Small, ContinuousAction -> False],
           Text@Style[Dynamic@padIt1[a, {3, 1}], 11], Spacer[2],
           Button[Text@Style["zero", 10], {a = 0.0; event = "reset"; gtick += del},
            ImageSize → {45, 20}, Alignment → Center],
           Button[Text@Style["one", 10], {a = 1.0; event = "reset"; gtick += del},
            ImageSize → {45, 20}, Alignment → Center]
          },
          {
           Text@Style["b", Italic, 12],
           Manipulator[Dynamic[b, {b = #; event = "reset"; gtick += del} &],
            {-10., 10., 0.1}, ImageSize → Small, ContinuousAction -> False,
            Enabled → Dynamic[forceTermSelection == 2 ||
```

```
        forceTermSelection == 4 || forceTermSelection == 5]],
   Text@Style[Dynamic@padIt1[b, {3, 1}], 11], Spacer[2],
   Button[Text@Style["zero", 10], {b = 0.0; event = "reset"; gtick += del},
    ImageSize → {45, 20}, Alignment → Bottom],
   Button[Text@Style["zero", 10], {b = 1.0; event = "reset"; gtick += del},
    ImageSize → {45, 20}, Alignment → Bottom]
  }
  ,
  {
   Text@Style["c", Italic, 12],
   Manipulator[Dynamic[c, {c = #; event = "reset"; gtick += del} &],
    {-10., 10., 0.1}, ImageSize → Small, ContinuousAction -> False,
    Enabled → Dynamic[forceTermSelection == 4 ||
        forceTermSelection == 5 || forceTermSelection == 2]],
   Text@Style[Dynamic@padIt1[c, {3, 1}], 11], Spacer[1],
   Button[Text@Style["zero", 10], {c = 0.0; event = "reset"; gtick += del},
    ImageSize → {45, 20}, Alignment → Bottom, BaselinePosition → Center],
   Button[Text@Style["zero", 10], {c = 1.0; event = "reset"; gtick += del},
    ImageSize → {45, 20}, Alignment → Bottom, BaselinePosition → Center]
  }
  ,
  {
   Text@Style[Style["x", Italic]_0, 12],
   Manipulator[Dynamic[x0, {x0 = #; event = "reset"; gtick += del} &],
    {-1.5, 1.5, 0.01}, ImageSize → Small, ContinuousAction -> False,
    Enabled → Dynamic[forceTermSelection == 3]],
   Text@Style[Dynamic@padIt1[x0, {3, 2}], 11], Spacer[1],
   Button[Text@Style["zero", 10], {x0 = 0.0; event = "reset"; gtick += del},
    ImageSize → {45, 20}, Alignment → Bottom, BaselinePosition → Center],
   Button[Text@Style["zero", 10], {x0 = 1.0; event = "reset"; gtick += del},
    ImageSize → {45, 20}, Alignment → Bottom, BaselinePosition → Center]
  }
  ,
  {
   Text@Style[Style["y", Italic]_0, 12],
   Manipulator[Dynamic[y0, {y0 = #; event = "reset"; gtick += del} &],
    {-1.5, 1.5, 0.01}, ImageSize → Small, ContinuousAction -> False,
    Enabled → Dynamic[forceTermSelection == 3]],
   Text@Style[Dynamic@padIt1[y0, {3, 2}], 11], Spacer[1],
   Button[Text@Style["zero", 10], {y0 = 0.0; event = "reset"; gtick += del},
    ImageSize → {45, 20}, Alignment → Bottom, BaselinePosition → Center],
   Button[Text@Style["zero", 10], {y0 = 1.0; event = "reset"; gtick += del},
    ImageSize → {45, 20}, Alignment → Bottom, BaselinePosition → Center]
  }
  ,
  {
   Item[
    Row[{
      Text@Style["σx", 12], Spacer[1],
      Manipulator[Dynamic[stdx, {stdx = #; event = "reset"; gtick += del} &],
       {0.1, 3, 0.05}, ImageSize → Tiny, ContinuousAction -> False,
       Enabled → Dynamic[forceTermSelection == 3]],
```

```
        Text@Style[Dynamic@padIt1[stdx, {3, 2}], 11],
        Spacer[30],
        Text@Style["σy", 12], Spacer[1],
        Manipulator[Dynamic[stdy, {stdy = #; event = "reset"; gtick += del} &],
         {0.1, 3, 0.05}, ImageSize → Tiny, ContinuousAction -> False,
         Enabled → Dynamic[forceTermSelection == 3]],
        Text@Style[Dynamic@padIt1[stdy, {3, 2}], 11]
       }], Alignment → Center
     ], SpanFromLeft
    }
   }, Spacings → {.12, .15}, Alignment → Center
  ]
 }
 ,
 {
  Dynamic[
   Block[{plotTitle, imageSize},
    imageSize = {ContentSizeW + 50, ContentSizeH - 250};
    plotTitle = Text@Style[Row[{Style["f", Italic], "(", Style["x", Italic],
         ", ", Style["y", Italic], ") = ", forceTermUsedFormatCommon[
          forceTermSelection, a, b, c, stdy, stdx, x0, y0, x, y]}], 11];

    Which[typeOfplotToShow == "ListPlot3D1" ||
      typeOfplotToShow == "ListPlot3D2" || typeOfplotToShow == "ListPlot3D3",
     (
      Grid[{
        {Panel[Plot3D[Evaluate@forceTermExpressionCommon[
             forceTermSelection, a, b, c, stdy, stdx, x0, y0, x, y],
           Evaluate@{x, grid[[-1, 1, 1]], grid[[-1, -1, 1]]},
           {y, grid[[-1, 1, 2]], grid[[1, 1, 2]]},
           PerformanceGoal → plotPerformanceGoal,
           ImagePadding → {{10, 10}, {20, 25}},
           ImageMargins → 1,
           PlotRange → All,
           PlotLabel → plotTitle,
           AxesLabel →
            {Text@Style["x", Italic, 11], Text@Style["y", Italic, 11], None},
           ImageSize → imageSize,
           TicksStyle → 9
          ], Alignment → Center, ImageMargins → 0, FrameMargins → 0]}},
       Spacings → {0, 0}, Frame → None
      ]
     ),

     typeOfplotToShow == "ArrayPlot",
     Panel[ArrayPlot[-forceGrid, ColorFunctionScaling -> True,
       ImageMargins → 1,
       ImageSize → imageSize,
       PlotLabel → plotTitle
      ], Alignment → Center, ImageMargins → 0, FrameMargins → 0],
```

```
          True,
          Panel[ListContourPlot[
            Flatten[MapThread[Append[#1, #2] &, {grid, -forceGrid}, 2], 1],
            Contours → 10,
            Evaluate[If[typeOfplotToShow == "ListContourPlot",
              ContourLabels → Function[{x, y, z}, Text[Framed[z], {x, y}]], {} ]],
            Frame → False,
            ImageMargins → 1,
            ImageSize → imageSize,
            PlotLabel -> plotTitle
           ], Alignment → Center, ImageMargins → 0, FrameMargins → 0]
         ]
        ]], SpanFromLeft
      }
     }, Spacings → {.1, .5}, Alignment → Center, Frame → None
    ], Alignment → {Center, Top}
   ],
  (*-----------------------------*)
  (*--- solver   macro      -----*)
  (*-----------------------------*)
  solverMacro = Item[Grid[{
      {
       Text@Style["initial solution", 12],
       RadioButtonBar[Dynamic[initialSolution, {initialSolution = #;
           event = "reset"; gtick += del} &], {"zero" -> Text@Style["zero", 11],
         "random" -> Text@Style["random", 11]}, ContinuousAction -> False]
      },
      {
       Style[Text["select solver"], 12],
       myGrid[{
         {
          RadioButtonBar[Dynamic[solverType, {solverType = #; event = "reset";
              gtick += del} &], {"direct" → Text@Style["direct sparse solver", 11],
            "stationary" → Grid[{{Text@Style["stationary", 11],
                PopupMenu[Dynamic[stationarySolver, {stationarySolver = #;
                    event = "reset" ; If[sorInputChoice == 1, SORomegaUserValue =
                       setOptimalSORomega[hx, hy, lenX, lenY]]; gtick += del} &],
                  {
                   "Jacobi" → Style["Jacobi", 11],
                   "Gauss-Seidel" → Style["Gauss-Seidel", 11],
                   "Gauss-Seidel red/black" → Style["Gauss-Seidel R/B", 11],
                   "SOR" → Style["SOR", 11],
                   "SOR/Chebyshev" → Style["SOR/Chebyshev", 11]
                  }, ImageSize -> All, ContinuousAction -> False,
                  Enabled → Dynamic[solverType == "stationary"]]}}],

            "non stationary" → Grid[{{Text@Style["nonstationary", 11],
                PopupMenu[Dynamic[nonStationarySolver,
                   {nonStationarySolver = #; event = "reset" ; gtick += del} &],
                  {
                   "steepest descent" → Style["steepest descent", 11],
                   "ConjugateGradient" → Style["conjugate gradient", 11],
```

```
              "GMRES" → Style["GMRES", 11],
              "BiCGSTAB" → Style["BiCGSTAB", 11]
             }, ImageSize -> All, ContinuousAction -> False,
             Enabled → Dynamic[solverType == "non stationary"]]}}]},
       ContinuousAction -> False, Appearance → "Vertical",
       BaselinePosition → Baseline]
      }
     }, Spacings → {.1, .1}]
    },
    {
     Style[Text[Column[{"stationary", "solvers", "configuration"}]], 12],

     Grid[{

       {RadioButtonBar[Dynamic[splittingOrRelaxation,
          {splittingOrRelaxation = #; event = "reset"; gtick += del} &],
         {
          "splitting matrix method" → Text@Style["splitting matrix method", 11],
          "relaxation method" -> Text@Style["relaxation method", 11]
         },
         Appearance → "Vertical",
         Enabled → Dynamic[solverType == "stationary"]]
       },
       {Grid[{
          {Text@Style["SOR ω value", 12], SpanFromLeft},
          {RadioButtonBar[Dynamic[sorInputChoice,
             {sorInputChoice = #; event = "reset", If[sorInputChoice == 1,
                SORomegaUserValue = setOptimalSORomega[hx, hy, lenX, lenY]];
               gtick += del} &], {1 → Text@Style["optimal", 11],
             2 → Text@Style["user", 11]
            }, Enabled →
             Dynamic[solverType == "stationary" && stationarySolver == "SOR"]],

           Manipulator[Dynamic[SORomegaUserValue,
             {SORomegaUserValue = #; event = "reset"; gtick += del} &],
            {0.01, 1.99, 0.01}, ImageSize -> Tiny, ContinuousAction -> False,
            Enabled → Dynamic[solverType == "stationary" && sorInputChoice == 2]],
           Text@Style[Dynamic@padIt2[SORomegaUserValue, {4, 2}], 11]
          }
         }, Alignment → Center ], SpanFromLeft
       }
      }, Spacings → {.1, .7}, Alignment → Left](*was LEFT*)
    },
    {
     Style[Text[Column[{"nonstationary", "solvers", "configuration"}]], 12],
     Grid[{
       {Style[Text["preconditioner"], 12],
        PopupMenu[Dynamic[preconditioner,
          {preconditioner = #; event = "reset" ; gtick += del} &],
         {
          "NONE" -> Style["NONE", 11],
          "SSOR" → Style["SSOR", 11],
```

```
            "ILU0" → Style[ "ILU0", 11],
            "ILUT" → Style[ "ILUT", 11],
            "ILUTP" →  Style[ "ILUTP", 11]

           }, ImageSize -> All, ContinuousAction -> False,
           Enabled → Dynamic[solverType == "non stationary"]], SpanFromLeft
         },
         {Style[Text["fill in"], 11],
          Manipulator[Dynamic[fillIn, {fillIn = #; event = "reset"; gtick += del} &],
           {0, 20, 1}, ImageSize -> Tiny, ContinuousAction -> False,
           Enabled → Dynamic[solverType == "non stationary" &&
              (preconditioner == "ILUT" || preconditioner == "ILUTP")]],
          Text@Style[Dynamic@padIt2[fillIn, {2, 0}], 11]
         }
        }, Alignment → Left]
     },
     {
      Text@Style[Column[{"convergence", "tolerance"}], 12],
      Grid[{
        {Spacer[2],
         Manipulator[Dynamic[toleranceConstant, {toleranceConstant = #;
             event = "reset"; gtick += del} &], {1, 6, 1}, ImageSize -> Tiny,
          ContinuousAction -> False, Enabled → Dynamic[Not[solverType == "direct"]]],
         Text@Style[Dynamic@padIt2[N@10^-toleranceConstant, {1, 6}], 12]
        }}, Spacings → {.5, 0}]
     }
    },

    (*configurations for the solver panel grid*)
    Alignment → {{Center, Center}},
    Spacings → {0, 1.4},
    Frame → All,
    FrameStyle -> Directive[Thickness[.005], Gray]
    (*ItemSize→{{13,23},Automatic}*)

   ], Alignment → {Center, Top}
  ]
 },
(*------------------------------*)
(*--- LEVEL 2              -----*)
(*------------------------------*)
With[{
  pde3 = Grid[{
     {TabView[{
        Style["geometry", 11] → geometryMacro,
        Style["solver", 11] → solverMacro,
        Style["source", 11] → sourceMacro
       }, ImageSize → {310, 410 }]
     }
    }, Spacings → {.1, 0}, Alignment → Center
   ]
```

```
    },
    (*--- end of level 2 ---*)

    ## &[
     Item[Grid[{
        {
         Grid[{
           {topRowMacro},
           {Graphics[Text[Style[Dynamic@gstatusMessage, 12]],
             ImageSize -> {240, 20}, ImageMargins → 0]}
          }, Spacings → {0, 0}], plotOptionsMacro}}, Spacings → {2.8, 0}
      ], ControlPlacement → Top ],
     Item[pde3, ControlPlacement → Left]
    ]
   ]
  ],
 (*----------- end of Manipulate controls ---------------------------*)

 {{gstatusMessage, "reseting..."}, None},
 {{gtick, 0}, None},
 {{del, 10 * $MachineEpsilon}, None},

 (*-- PDE 3 parameters ---*)
 {{state, "INIT"}, None}, (*4 states*)
 {{event, "reset"}, None},
 {{stepNumber, 0}, None},
 {{mask, {}}, None},
 {{cpuTimeUsed, 0}, None},
 {{fillIn, 8}, None},

 (*these are the A=M-N, for splitting method. Save M^-1 since that is what is used*)
 {{Minv, {}}, None},
 {{splittingNmatrix, {}}, None},

 (*this is the A=L+D+U factorization*)
 {{factorLmatrix, {}}, None},
 {{factorDmatrix, {}}, None},
 {{factorUmatrix, {}}, None},

 (*This is the error iteration matrix T=(M^-1).N*)
 {{iterationMatrix, {}}, None},
 {{nonStationarySolver, "ConjugateGradient"}, None},
 {{stationarySolver, "Jacobi"}, None},
 {{solverType, "stationary"}, None},
 {{splittingOrRelaxation, "relaxation method"}, None},
 {{nRow, 0}, None},
 {{preConditionerMatrix, {}}, None},
 {{preconditioner, "NONE"}, None},
 {{finalDisplayImage, {}}, None},
 {{rightHandVector, {}}, None},
 {{systemAmatrix, {}}, None},
 {{steepestDescentR, {}}, None},
```

```
{{conjugateGradientR, {}}, None},
{{conjugateGradientP, {}}, None},
{{conjugateGradientZ, {}}, None},
{{addFaceGrids, False}, None},
{{zAxisScale, False}, None},
{{plotPerformanceGoal, "Quality"}, None},
{{hx, 1 / 4}, None},
{{hy, 1 / 4}, None},
{{lenX, 1}, None},
{{lenY, 1}, None},
{{centerGrid, True}, None},
{{a, 1.0}, None},
{{b, 0.0}, None},
{{c, 0.0}, None},
{{x0, 0.0}, None},
{{y0, 0.0}, None},
{{stdx, 0.3}, None},
{{stdy, 0.3}, None},
{{residualPlotData, Table[0., {10 000}]}, None},
{{initialSolution, "zero"}, None},
{{forceConstant, 1}, None},
{{forceTermSelection, 1}, None},
{{sorInputChoice, 1}, None},
{{SORomegaUserValue, 1.9}, None},
{{SORChebyomega, 1.0}, None},
{{typeOfplotToShow, "ListPlot3D1"}, None},
{{plotToShow, "solution"}, None},
{{northBCtype, "Dirichlet"}, None},
{{northbc, (1) &}, None},
{{northBCconstantValue, 0}, None},
{{westBCtype, "Dirichlet"}, None},
{{westbc, (1) &}, None},
{{westBCconstantValue, 0}, None},
{{eastBCtype, "Dirichlet"}, None},
{{eastbc, (1) &}, None},
{{eastBCconstantValue, 0}, None},
{{southBCtype, "Dirichlet"}, None},
{{southbc, (1) &}, None},
{{southBCconstantValue, 0}, None},
{{relativeResidual, 1}, None},
{{toleranceConstant, 6}, None},
{{residual, {}}, None},
{{residualPlotLimits, {0, 0.1}}, None},
{{u, Table[0, {5}, {5}]}, None},
{{forceGrid, Table[-1, {5}, {5}]}, None},
{{grid, makeGridCommon[0.25, 0.25, 1, 1, True]}, None},
{{normf, 0}, None},
{{cpuTimeUsed, 0}, None},
{{stepNumber, 0}, None},

ControlPlacement → Left,
SynchronousUpdating → False,
```

```
 SynchronousInitialization → True,
 ContinuousAction → False,
 Alignment -> Center,
 ImageMargins → 0,
 FrameMargins → 0,
 TrackedSymbols :→ {gtick},
 Paneled → True,
 Frame → False,
 Initialization :→
  {

   evaluateForceCommon = Unevaluated[#3 #1^#6 + #4 #2^#7] /. HoldPattern[0.0^0.0] :→ 0.0 &;
   (*-------------------------------------------------*)
   padIt1[v_?(NumericQ[#] && Im[#] == 0 &), f_List] := AccountingForm[Chop[N@v] , f,
     NumberSigns -> {"-", "+"}, NumberPadding -> {"0", "0"}, SignPadding -> True];
   (*-------------------------------------------------*)
   padIt2[v_?(NumericQ[#] && Im[#] == 0 &), f_List] := AccountingForm[Chop[N@v] , f,
     NumberSigns -> {"", ""}, NumberPadding -> {"0", "0"}, SignPadding -> True];
   (*-------------------------------------------------*)
   padIt6[v_?(NumericQ[#] && Im[#] == 0 &), f_] :=
    AccountingForm[v, f, NumberSigns -> {"", ""},
     NumberPadding -> {"0", "0"}, SignPadding -> True, NumberPoint → ""];
   (*-------------------------------------------------*)
getSolutionDomainDimensions[hx_, hy_, Lx_, Ly_] := Module[{nRow, nCol},
     nRow = Ly / hy + 1;
     nCol = Lx / hx + 1;
     {nRow, nCol}
    ];
(*-------------------------------------------------*)
   generateEigenvalues[systemMatrix_, preConditionerMatrix_] :=
    Module[{eigs, conditionedEigs, maximumEig, condA, condMA},
     eigs = Norm /@ Eigenvalues[systemMatrix];
     condA = LUDecomposition[systemMatrix][[3]];
     condMA = LUDecomposition[preConditionerMatrix.systemMatrix][[3]];

     conditionedEigs = Norm /@ Eigenvalues[Normal[preConditionerMatrix.systemMatrix]];
     maximumEig = Max[{eigs, conditionedEigs}];

     {eigs, conditionedEigs, maximumEig, condA, condMA}
    ];
   (*-------------------------------------------------*)
makeGridCommon[hx_, hy_, Lx_, Ly_, centerGrid_] := Module[{i, j, nx, ny, grid},
     nx = Lx / hx + 1;
     ny = Ly / hy + 1;

     With[{$icfrom = Floor[ny / 2], $icto = -Floor[ny / 2],
       $jcfrom = -Floor[nx / 2], $jcto = Floor[nx / 2], $ifrom = ny - 1, $jto = nx - 1},
      grid = If[centerGrid,
        Table[{j * hx, i * hy}, {i, $icfrom, $icto, -1}, {j, $jcfrom, $jcto}],
        Table[{j * hx, i * hy}, {i, $ifrom, 0, -1}, {j, 0, $jto}]
       ]
```

```
      ];

      N@grid
     ];
(*-------------------------------------------------*)
    forceTermUsedFormatCommon[forceTermSelection_, a_,
      b_, c_, stdy_, stdx_, x0_, y0_, x_, y_] := Module[{forceTermUsed},
      Which[
       forceTermSelection == 1, forceTermUsed = a ,
       forceTermSelection == 2, forceTermUsed = HoldForm[ x y],
       forceTermSelection == 3, forceTermUsed =
        HoldForm[a Exp[- ( (x - x0) ^2 / (2 (stdx) ^2) + (y - y0) ^2 / (2 (stdy) ^2)) ]],
       forceTermSelection == 4, forceTermUsed = HoldForm[a (Cos[b Pi x] + Sin[c Pi y])],
       forceTermSelection == 5, forceTermUsed = HoldForm[a (Cos[b Pi x] * Sin[c Pi y])]
      ];
      forceTermUsed
     ];
    (*-------------------------------------------------*)
forceTermExpressionCommon[forceTermSelection_, a_,
      b_, c_, stdy_, stdx_, x0_, y0_, x_, y_] := Module[{forceTermUsed},
      Which[
       forceTermSelection == 1, forceTermUsed = a ,
       forceTermSelection == 2, forceTermUsed = x y,
       forceTermSelection == 3,
       forceTermUsed = a Exp[- ( (x - x0) ^2 / (2 (stdx) ^2) + (y - y0) ^2 / (2 (stdy) ^2)) ],
       forceTermSelection == 4, forceTermUsed = a (Cos[b Pi x] + Sin[c Pi y]),
       forceTermSelection == 5, forceTermUsed = a (Cos[b Pi x] * Sin[c Pi y])
      ];

      forceTermUsed
     ];
    (*-------------------------------------------------*)
    isInternalNode[i_, j_, nx_, ny_] := If[(i > 1 && i < ny && j > 1 && j < nx), True, False];
    (*-------------------------------------------------*)
    isEdgeCommon[i_, j_, nCol_, nRow_] := If[((i == 1 || i == nRow) && (j > 1 && j < nCol)) ||
       ((j == 1 || j == nCol) && (i > 1 && i < nRow)), True, False];
    (*-------------------------------------------------*)
    setCornerNodeCommon[$u_, i_, j_, nx_, ny_] := Module[{u = $u},
      Which[
       i == 1 && j == 1, u[[1, 1]] = Mean[{ u[[2, 1 ]], u[[1, 2]]}],
       i == 1 && j == nx, u[[1, nx]] = Mean[{ u[[1, nx - 1]], u[[2, nx ]]}],
       i == ny && j == 1, u[[ny, 1]] = Mean[{ u[[ny - 1, 1 ]], u[[ny, 2]]}],
       i == ny && j == nx, u[[ny, nx]] = Mean[{ u[[ny, nx - 1]], u[[ny - 1, nx ]]}]
      ];
      u
     ];
    (*-------------------------------------------------*)
makeScrolledPaneCommon[mat_?(MatrixQ[#, NumberQ] &), nRow_?(IntegerQ[#] && Positive[#] &),
      nCol_?(IntegerQ[#] && Positive[#] &)] := Module[{t},
      t = Grid[mat, Spacings → {.4, .4}, Alignment → Left, Frame → All];
      t = Text@Style[NumberForm[Chop[N@t] , {6, 5}, NumberSigns -> {"-", ""},
          NumberPadding -> {"", ""}, SignPadding -> True], LineBreakWithin → False];
```

```
  Pane[t, ImageSize → {nCol, nRow}, Scrollbars → True]
 ];
(*-----------------------------------------------*)
makeScrolledPaneCommon[lst_?(VectorQ[#, NumericQ] &), nRow_?
   (IntegerQ[#] && Positive[#] &), nCol_?(IntegerQ[#] && Positive[#] &)] := Module[{t},
  t = Grid[{lst}, Spacings → {.4, .4}, Alignment → Left, Frame → All];
  t = Text@Style[AccountingForm[Chop[N@t] , {6, 5}, NumberSigns -> {"-", ""},
      NumberPadding -> {"", ""}, SignPadding -> True], LineBreakWithin → False];
  Pane[t, ImageSize → {nCol, nRow}, Scrollbars → True]
 ];
(*-----------------------------------------------*)
processCornersCommon[u_, $eqs_, n_, U_, i_, j_, $vars_, northBCtype_, westBCtype_,
  southBCtype_, eastBCtype_, nRow_, nCol_] := Module[{vars = $vars, eqs = $eqs},

  vars⟦n⟧ = U⟦i, j⟧;

  Which[i == 1 && j == 1, (*top left-top corner*)
   (
    Which[northBCtype == "Dirichlet" || westBCtype == "Dirichlet",
     (
      If[northBCtype == "Dirichlet" && westBCtype == "Dirichlet",
       (
        eqs⟦n⟧ = U⟦i, j⟧ == Mean[{u⟦1, 2⟧, u⟦2, 1⟧}]
       ),
       (
        eqs⟦n⟧ = U⟦i, j⟧ == u⟦1, 1⟧
       )
      ]
     ), True, (*both edgs are not Dirichlet*)
     (
      eqs⟦n⟧ = U⟦i, j⟧ - Mean[{U⟦1, 2⟧, U⟦2, 1⟧}] == 0
     )
    ]
   ),
   i == nRow && j == 1, (*bottom left corner*)
   (
    Which[southBCtype == "Dirichlet" || westBCtype == "Dirichlet",
     (
      If[southBCtype == "Dirichlet" && westBCtype == "Dirichlet",
       eqs⟦n⟧ = U⟦i, j⟧ - Mean[{u⟦i, 2⟧, u⟦i - 1, 1⟧}] == 0,
       eqs⟦n⟧ = U⟦i, j⟧ == u⟦nRow, 1⟧
      ]
     ), True, (*both edgs are not Dirichlet*)
     (
      eqs⟦n⟧ = U⟦i, j⟧ - Mean[{U⟦i, 2⟧, U⟦i - 1, 1⟧}] == 0.0
     )
    ]
   ), (*top right corner*)
   i == 1 && j == nCol,
   (
    Which[northBCtype == "Dirichlet" || eastBCtype == "Dirichlet",
     (
```

```
        If[northBCtype == "Dirichlet" && eastBCtype == "Dirichlet",
         eqs[[n]] = U[[i, j]] - Mean[{U[[1, j - 1]], U[[2, j]]}] == 0.0,
         eqs[[n]] = U[[i, j]] == u[[1, nCol]]
        ]
       ), True, (*both edgs are not Dirichlet*)
       (
        eqs[[n]] = U[[i, j]] - Mean[{U[[1, j - 1]], U[[2, j]]}] == 0.0;
       )
      ]
     ),
     i == nRow && j == nCol, (*both edgs are not Dirichlet*)
     (
      Which[southBCtype == "Dirichlet" || eastBCtype == "Dirichlet",
       (
        If[southBCtype == "Dirichlet" && eastBCtype == "Dirichlet",
         eqs[[n]] = U[[i, j]] - Mean[{U[[i, j - 1]], U[[i - 1, j]]}] == 0.0,
         eqs[[n]] = U[[i, j]] - u[[nRow, nCol]]
        ]
       ), True, (*both edgs are nuemman*)
       (
        eqs[[n]] = U[[i, j]] - Mean[{U[[i, j - 1]], U[[i - 1, j]]}] == 0.0
       )
      ]
     )
    ];

    {eqs, vars}
   ];
(*-----------------------------------------------------*)
  processPDE[$u_, $state_, event_, $forceGrid_, $grid_, $normf_, $cpuTimeUsed_,
    $stepNumber_, $mask_, $rightHandVector_, $Minv_, $NN_, $factorLmatrix_,
    $factorDmatrix_, $factorUmatrix_, $iterationMatrix_, $A_, $steepestDescentR_,
    $conjugateGradientR_, $conjugateGradientP_, $conjugateGradientZ_, residualPlotData_,
    (*by reference*)SORomegaUserValue_, $SORChebyomega_, $residualPlotLimits_,
    $preConditionerMatrix_, $relativeResidual_, $residual_, fillIn_,
    nonStationarySolver_, stationarySolver_, solverType_, splittingOrRelaxation_,
    preconditioner_, addFaceGrids_, zAxisScale_, plotPerformanceGoal_, hx_, hy_,
    Lx_, Ly_, centerGrid_, a_, b_, c_, x0_, y0_, stdx_, stdy_, initialSolution_,
    forceTermSelection_, typeOfplotToShow_, plotToShow_, northBCtype_,
    northbc_, northBCconstantValue_, westBCtype_, westbc_, westBCconstantValue_,
    eastBCtype_, eastbc_, eastBCconstantValue_, southBCtype_, southbc_,
    southBCconstantValue_, toleranceConstant_, $finalDisplayImage_, del_, gtick_] :=
   Module[{successInitialization = True, makeNewPlot = True, u = $u, state = $state,
     forceGrid = $forceGrid, grid = $grid, normf = $normf, cpuTimeUsed = $cpuTimeUsed,
     stepNumber = $stepNumber, mask = $mask, rightHandVector = $rightHandVector,
     Minv = $Minv, NN = $NN, factorLmatrix = $factorLmatrix, factorDmatrix = $factorDmatrix,
     factorUmatrix = $factorUmatrix, iterationMatrix = $iterationMatrix, AA = $A,
     steepestDescentR = $steepestDescentR, conjugateGradientR = $conjugateGradientR,
     conjugateGradientP = $conjugateGradientP, conjugateGradientZ = $conjugateGradientZ,
     relativeResidual = $relativeResidual, residual = $residual,
     SORChebyomega = $SORChebyomega, residualPlotLimits = $residualPlotLimits,
     preConditionerMatrix = $preConditionerMatrix, finalDisplayImage = $finalDisplayImage,
```

```
 tmpU, tmpGrid, tmpForceGrid, tick = gtick, statusMessage = ""
},
(*----- INIT STATE -----*)
Which[state == "INIT",
 (
  {u, SORChebyomega, relativeResidual, grid, forceGrid,
    residual, residualPlotLimits, normf, mask, cpuTimeUsed, stepNumber,
    successInitialization, statusMessage, AA, rightHandVector, factorLmatrix,
    factorDmatrix, factorUmatrix, preConditionerMatrix, Minv,
    NN, iterationMatrix} = initializeSystem[
    nonStationarySolver, preconditioner, splittingOrRelaxation, northBCtype, northbc,
    northBCconstantValue, westBCtype, westbc, westBCconstantValue, eastBCtype,
    eastbc, eastBCconstantValue, southBCtype, southbc, southBCconstantValue, hx,
    hy, Lx, Ly, solverType, stationarySolver, centerGrid, forceTermSelection,
    a, b, c, x0, y0, stdx, stdy, initialSolution, SORomegaUserValue];

  (*-- Only switch to new state if initialization is success --*)
  If[successInitialization,
   (
    makeNewPlot = True;

    Which[event == "reset", statusMessage = "reset complete",
     event == "run_button",
     (
      state = "RUNNING";
      tick += del
     ),
     event == "pause_button",
     (
      state = "PAUSE";
      tick += del
     ),
     event == "step_button",
     (
      state = "RUNNING";
      tick += del
     )
    ]
   ),
   makeNewPlot = False
  ]
 ),
 (*----- PAUSE STATE -----*)
 state == "PAUSE",
 (
  statusMessage = Row[{"paused [", stepNumber, "]"}];

  Which[
   event == "pause_button",
   (
    state = "PAUSE"
   ),
```

```
    event == "reset",
    (
     state = "INIT";
     {tmpU, SORChebyomega, relativeResidual, tmpGrid, tmpForceGrid,
       residual, residualPlotLimits, normf, mask, cpuTimeUsed, stepNumber,
       successInitialization, statusMessage, AA, rightHandVector, factorLmatrix,
       factorDmatrix, factorUmatrix, preConditionerMatrix, Minv,
       NN, iterationMatrix} = initializeSystem[
       nonStationarySolver, preconditioner, splittingOrRelaxation,
       northBCtype, northbc, northBCconstantValue, westBCtype, westbc,
       westBCconstantValue, eastBCtype, eastbc, eastBCconstantValue,
       southBCtype, southbc, southBCconstantValue, hx, hy, Lx, Ly,
       solverType, stationarySolver, centerGrid, forceTermSelection, a,
       b, c, x0, y0, stdx, stdy, initialSolution, SORomegaUserValue];
     If[successInitialization,
      {u, grid, forceGrid} = {tmpU, tmpGrid, tmpForceGrid},
      makeNewPlot = False
     ];

     tick += del
    ),
    event == "run_button" || event == "step_button",
    (
     state = "RUNNING";
     tick += del
    )
   ]
  ),
  (*----- RUNNING STATE -----*)
  state == "RUNNING",
  (
   Which[event == "step_button" || event == "run_button",
    (
     If[stepNumber == 0 || relativeResidual > 10^(-toleranceConstant),
      (
       {u, relativeResidual, residual, steepestDescentR, conjugateGradientR,
         conjugateGradientZ, conjugateGradientP, SORChebyomega, cpuTimeUsed} =
        solve[u, AA, rightHandVector, mask, hx, hy, relativeResidual,
         residual, stationarySolver, solverType, splittingOrRelaxation,
         cpuTimeUsed, nonStationarySolver, Minv, NN, normf, forceGrid,
         SORomegaUserValue, stepNumber, steepestDescentR, conjugateGradientR,
         conjugateGradientZ, preconditioner, preConditionerMatrix,
         fillIn, toleranceConstant, SORChebyomega, conjugateGradientP,
         northBCtype, northbc, northBCconstantValue, westBCtype, westbc,
         westBCconstantValue, eastBCtype, eastbc, eastBCconstantValue,
         southBCtype, southbc, southBCconstantValue, grid];

       stepNumber = stepNumber + 1;
       statusMessage = Row[{"running [", IntegerPart[stepNumber], "]"}];
       residualPlotData[[stepNumber]] = relativeResidual;

       (*-- only re-loop if in running state --*)
```

```
        If[event == "run_button",
         tick += del,
         state = "PAUSE"(*082713*)
        ]
       ),
       (
        statusMessage = Row[{"completed [", stepNumber, "]"}];
       )
      ]
     ),
     event == "reset",
     (
      state = "INIT";
      {tmpU, SORChebyomega, relativeResidual, tmpGrid,
        tmpForceGrid, residual, residualPlotLimits, normf, mask, cpuTimeUsed,
        stepNumber, successInitialization, statusMessage, AA,
        rightHandVector, factorLmatrix,
        factorDmatrix, factorUmatrix, preConditionerMatrix, Minv,
        NN, iterationMatrix} = initializeSystem[
        nonStationarySolver, preconditioner, splittingOrRelaxation,
        northBCtype, northbc, northBCconstantValue, westBCtype, westbc,
        westBCconstantValue, eastBCtype, eastbc, eastBCconstantValue,
        southBCtype, southbc, southBCconstantValue, hx, hy, Lx, Ly,
        solverType, stationarySolver, centerGrid, forceTermSelection, a,
        b, c, x0, y0, stdx, stdy, initialSolution, SORomegaUserValue];
      If[successInitialization,
       {u, grid, forceGrid} = {tmpU, tmpGrid, tmpForceGrid},
       makeNewPlot = False
      ];

      tick += del
     ),
     event == "pause_button",
     (
      state = "PAUSE";
      tick += del
     )
    ]
  )
 ];

(* state machine completed, plot the final result *)
If[makeNewPlot,
 finalDisplayImage =
  If[plotToShow == "solution data" || plotToShow == "system matrix information",
   Grid[{makeFinalPlot[u, grid, Lx, Ly, plotToShow, typeOfplotToShow,
      plotPerformanceGoal, addFaceGrids, zAxisScale, AA, residual,
      residualPlotLimits]}, Alignment → Center, Spacings → {0, 0}
   ]
   ,
   If[plotToShow == "convergence",
    Grid[{
```

```
        {makeResidualPlot[stepNumber,
          residualPlotData, {ContentSizeW, ContentSizeH - 8}, 1.2, {
           {None, None},
           {Text@Style["step number", 11],
            Text@Style[Grid[{
                 {"current residual",
                  NumberForm[relativeResidual, {11, 9}], SpanFromLeft},
                 {"cpu time", NumberForm[cpuTimeUsed, {7, 5}] , "sec"}
                }, Alignment → Center, Spacings → {.2, .3}
               ], 11]}
          }]
        }
       }, Alignment → Center, Spacings → {0, 0.2},
       Frame → None, FrameStyle -> Directive[Thickness[.005], Gray]
      ]
      ,
      Grid[{
        {makeResidualPlot[stepNumber,
          residualPlotData, {ContentSizeW, ContentSizeH - 285}, 0.2, {
           {None, None},
           {Text@Style["step number", 11],
            Text@Style[Grid[{
                 {"residual ", NumberForm[relativeResidual, {10, 8}],
                  "cpu ", NumberForm[cpuTimeUsed, {7, 5}]}}, Alignment → Left
               ], 11]}
          }]},
        makeFinalPlot[u, grid,
         Lx, Ly, plotToShow, typeOfplotToShow, plotPerformanceGoal,
         addFaceGrids, zAxisScale, AA, residual, residualPlotLimits]
       }, Alignment → Center, Spacings → {0, 0.2}, Frame → None,
       FrameStyle -> Directive[Thickness[.005], Gray]
      ]
     ]
    ]
   ];

   {finalDisplayImage, u, state, forceGrid, grid, normf, cpuTimeUsed, stepNumber, mask,
    Minv, NN, factorLmatrix, factorDmatrix, factorUmatrix, iterationMatrix, AA,
    rightHandVector, steepestDescentR, conjugateGradientR, conjugateGradientP,
    conjugateGradientZ, residualPlotData, relativeResidual, residual,
    preConditionerMatrix, residualPlotLimits, SORChebyomega, statusMessage, tick}
  ];
 (*-----------------------------------------------*)
 makeResidualPlot[stepNumber_,
   residualPlotData_, imageSize_, aspectRatio_, label_] := Module[{},
   Show[
    ListPlot[If[stepNumber == 0, {0}, residualPlotData⟦1 ;; stepNumber⟧],
     ImageSize → imageSize,
     Joined → True,
     Frame → True,
     AspectRatio → aspectRatio,
     FrameLabel → label,
```

```
      ImagePadding → {{30, 10}, {35, 40}},
      GridLines → Automatic,
      PlotStyle → {Gray, Thin},
      PlotRange →
       {All, {0, 1.1 * If[stepNumber == 0, 1, Max[residualPlotData[[1 ;; stepNumber + 1]]]]}},
      TicksStyle → 8
     ],
     ListPlot[If[stepNumber == 0, {0}, residualPlotData[[1 ;; stepNumber]]],
      PlotStyle → Red,
      PlotRange → {All, {0, 1.1 * Max[residualPlotData[[1 ;; stepNumber + 1]]]}}]
    ]
   ];
  (*-------------------------------------------------*)
  setUnknownsMask[{nRow_, nCol_}, northBCtype_,
    westBCtype_, eastBCtype_, southBCtype_] := Module[{mask},
    (*
    there are 7 cases to check for. Let T=top edge, R=right edge,
    L=left edge B=bottom edge, then we need to check for one of these
     cases: no Nuemman on any edge, {LTRB},{TL,TR,TB},{TLR,TBR,TBL},{LB},{LR},{LBR},
    {BR}. mask is used to tell location on unknowns in the solution grid. This
     is needed since now we have Nuemman BC and so nodes on the edge can be
     part of the unknown and we need later to find the location of the unknowns
    *)
    mask = Table[0, {nRow}, {nCol}];
    Which[westBCtype == "Dirichlet" && northBCtype == "Dirichlet" &&
      eastBCtype == "Dirichlet" && southBCtype == "Dirichlet",
     mask[[2 ;; -2, 2 ;; -2]] = 1,
     northBCtype == "Neumann" && westBCtype == "Dirichlet" &&
      eastBCtype == "Dirichlet" && southBCtype == "Dirichlet",
     mask[[1 ;; -2, 2 ;; -2]] = 1,
     northBCtype == "Dirichlet" && westBCtype == "Neumann" &&
      eastBCtype == "Dirichlet" && southBCtype == "Dirichlet",
     mask[[2 ;; -2, 1 ;; -2]] = 1,
     northBCtype == "Dirichlet" && westBCtype == "Dirichlet" &&
      eastBCtype == "Neumann" && southBCtype == "Dirichlet",
     mask[[2 ;; -2, 2 ;; -1]] = 1,
     northBCtype == "Dirichlet" && westBCtype == "Dirichlet" &&
      eastBCtype == "Dirichlet" && southBCtype == "Neumann",
     mask[[2 ;; -1, 2 ;; -2]] = 1,
     (* now do the checks for {TL,TR,TB} case*)
     northBCtype == "Neumann" && westBCtype == "Neumann" &&
      eastBCtype == "Dirichlet" && southBCtype == "Dirichlet",
     mask[[1 ;; -2, 1 ;; -2]] = 1,
     northBCtype == "Neumann" && westBCtype == "Dirichlet" &&
      eastBCtype == "Neumann" && southBCtype == "Dirichlet",
     mask[[1 ;; -2, 2 ;; -1]] = 1,
     northBCtype == "Neumann" && westBCtype == "Dirichlet" &&
      eastBCtype == "Dirichlet" && southBCtype == "Neumann",
     mask[[1 ;; -1, 2 ;; -2]] = 1,
     northBCtype == "Neumann" && westBCtype == "Neumann" &&
      eastBCtype == "Neumann" && southBCtype == "Dirichlet",
     mask[[1 ;; -2, 1 ;; -1]] = 1,
```

```
        northBCtype == "Neumann" && westBCtype == "Dirichlet" &&
          eastBCtype == "Neumann" && southBCtype == "Neumann",
        mask[[1 ;; -1, 2 ;; -1]] = 1,
        northBCtype == "Neumann" && westBCtype == "Neumann" &&
          eastBCtype == "Dirichlet" && southBCtype == "Neumann",
        mask[[1 ;; -1, 1 ;; -2]] = 1,
        northBCtype == "Dirichlet" && westBCtype == "Neumann" &&
          eastBCtype == "Dirichlet" && southBCtype == "Neumann",
        mask[[2 ;; -1, 1 ;; -2]] = 1,
        northBCtype == "Dirichlet" && westBCtype == "Neumann" &&
          eastBCtype == "Neumann" && southBCtype == "Dirichlet",
        mask[[2 ;; -2, All]] = 1,
        northBCtype == "Dirichlet" &&
          westBCtype == "Neumann" && eastBCtype == "Neumann" && southBCtype == "Neumann",
        mask[[2 ;; -1, All]] = 1,
        northBCtype == "Dirichlet" && westBCtype == "Dirichlet" &&
          eastBCtype == "Neumann" && southBCtype == "Neumann",
        mask[[2 ;; -1, 2 ;; -1]] = 1
      ];

      mask
     ];
(*-------------------------------------------------*)
   directSolver[$u_, AA_, rightHandVector_, mask_] :=
    Module[{loc, x, u = $u, relativeResidual, residual, nRow, nCol},

     {nRow, nCol} = Dimensions[u];
     x = LinearSolve[N@AA, rightHandVector];
     loc = Position[mask, 1];
     MapThread[(u = ReplacePart[u, #1 → #2 ]) &, {loc, x}];
     relativeResidual = 0.;
     With[{$nRow = nRow, $nCol = nCol},
      residual = Table[0, {$nRow}, {$nCol}]
     ];

     {u, relativeResidual, residual}
    ];
   (*-------------------------------------------------*)
steepestDescentSolve[$u_, $residual_, $steepestDescentR_,
     rightHandVector_, hx_, hy_, AA_, mask_, stepNumber_] :=
    Module[{u = $u, residual = $residual, steepestDescentR = $steepestDescentR,
      w, alpha, loc, res, nRow, nCol, relativeResidual},

     {nRow, nCol} = Dimensions[u];
     loc = Position[mask, 1];
     If[stepNumber == 0,
      (
       steepestDescentR = rightHandVector - AA.Flatten[Extract[u, Position[mask, 1]]]
      )
     ];

     w = AA.steepestDescentR;
```

```
      alpha = Norm[steepestDescentR]^2 / (steepestDescentR.w);
      With[{$nRow = nRow, $nCol = nCol},
       res = Table[0, {$nRow}, {$nCol}]
      ];

      (*convert the residue vector to matrix*)
      MapThread[(res = ReplacePart[res, #1 → #2 ]) &, {loc, steepestDescentR}];

      u = u + alpha * res;
      steepestDescentR = steepestDescentR - alpha * w;
      relativeResidual = (hx * hy) ^ (1 / 4) * Norm[steepestDescentR];
      residual = res;

      {u, relativeResidual, residual, steepestDescentR}
     ];
(*-------------------------------------------------*)
   krylovSolver[$u_, AA_, rightHandVector_,
      nonStationarySolver_, preconditioner_, toleranceConstant_, fillIn_, mask_] :=
     Module[{u = $u, x, loc, nRow, nCol, relativeResidual, residual},

      {nRow, nCol} = Dimensions[u];

      Which[preconditioner == "ILU0",
       x = LinearSolve[AA, rightHandVector, Method →
          {"Krylov", Method -> nonStationarySolver, "Preconditioner" → preconditioner,
           MaxIterations → Automatic, Tolerance → 10^-toleranceConstant}],
       True,
       x = LinearSolve[AA, rightHandVector, Method -> {"Krylov", Method ->
            nonStationarySolver, "Preconditioner" → {preconditioner, "FillIn" → fillIn},
           MaxIterations → Automatic, Tolerance → 10^-toleranceConstant}]
      ];

      loc = Position[mask, 1];
      MapThread[(u = ReplacePart[u, #1 → #2 ]) &, {loc, x}];

      relativeResidual = 0.;
      With[{$nRow = nRow, $nCol = nCol},
       residual = Table[0, {$nRow}, {$nCol}]
      ];

      {u, relativeResidual, residual}
     ];
(*-------------------------------------------------*)
   conjugateGradientSolve[$u_, $conjugateGradientR_,
      $conjugateGradientZ_, $conjugateGradientP_, $residual_, AA_,
      rightHandVector_, mask_, stepNumber_, preConditionerMatrix_, hx_, hy_] :=
     Module[{u = $u, residual = $residual, conjugateGradientR = $conjugateGradientR,
       conjugateGradientZ = $conjugateGradientZ, conjugateGradientP = $conjugateGradientP,
       w, alpha, beta, oldZ, oldR, resP, nRow, nCol, loc, relativeResidual},

      {nRow, nCol} = Dimensions[u];
      loc = Position[mask, 1];
```

```
    If[stepNumber == 0,
     (
      conjugateGradientR = rightHandVector - AA.Flatten[Extract[u, Position[mask, 1]]];
      conjugateGradientZ = preConditionerMatrix.conjugateGradientR;
      conjugateGradientP = conjugateGradientZ;
     )
    ];

    w = AA.conjugateGradientP;
    alpha = (conjugateGradientR.conjugateGradientZ) / (conjugateGradientP.w);

    With[{$nRow = nRow, $nCol = nCol},
     resP = Table[0, {$nRow}, {$nCol}]
    ];

    (*convert the residue vector to matrix*)
    MapThread[(resP = ReplacePart[resP, #1 → #2 ]) &, {loc, conjugateGradientP}];
    u = u + alpha * resP;
    oldR = conjugateGradientR;
    oldZ = conjugateGradientZ;
    conjugateGradientR = conjugateGradientR - alpha * w;

    conjugateGradientZ = preConditionerMatrix.conjugateGradientR;
    beta = (conjugateGradientZ.conjugateGradientR) / (oldZ.oldR);
    conjugateGradientP = conjugateGradientZ + beta * conjugateGradientP;
    relativeResidual = 1.0 * (hx * hy) ^ (1 / 4) * Norm[conjugateGradientR];
    residual = resP;

    {u, relativeResidual, residual,
     conjugateGradientR, conjugateGradientZ, conjugateGradientP}
];
  (*-----------------------------------------------*)
  iterativeSolveJacobi[$u_, $residual_, forceGrid_,
    hx_, hy_, normf_, northBCtype_, northbc_, northBCconstantValue_,
    westBCtype_, westbc_, westBCconstantValue_, eastBCtype_, eastbc_,
    eastBCconstantValue_, southBCtype_, southbc_, southBCconstantValue_, grid_] :=
   Module[{uNew = $u, u = $u, residual = $residual, sum = 1.0 * (hx^2 + hy^2),
     product = 1.0 * hx^2 * hy^2, i, j, nRow, nCol, relativeResidual},

    {nRow, nCol} = Dimensions[u];
    For[i = 1, i ≤ nRow, i++,
     (

      For[j = 1, j ≤ nCol, j++,
       (

        If[isEdgeCommon[i, j, nCol, nRow],
         (
          uNew = updateInternalEdgeNode[uNew, i, j, hx,
```

```
          hy, northBCtype, northBCconstantValue, northbc, southBCtype,
          southBCconstantValue, southhbc, westBCtype, westBCconstantValue,
          westbc, eastBCtype, eastBCconstantValue, eastbc, grid, forceGrid]
        ),
        (
         If[isInternalNode[i, j, nCol, nRow],
          (
           uNew[[i, j]] = 1/(2 sum) (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) +
               hx^2 (u[[i - 1, j]] + u[[i + 1, j]]) + product * forceGrid[[i, j]]);
           residual[[i, j]] = forceGrid[[i, j]] + (1/product (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) +
                  hx^2 (u[[i - 1, j]] + u[[i + 1, j]]) - 2 * sum * u[[i, j]]))
          ),
          (
           uNew = setCornerNodeCommon[uNew, i, j, nCol, nRow]
          )
         ](*ENDIF*)
        )
       ](*ENDIF*)
      )
     ](*END FOR*)
    )
   ]; (*END FOR*)
   u = uNew;
   relativeResidual = getRelativeResidual[normf, residual, hx, hy];
   {u, relativeResidual, residual}
  ];
(*--------------------------------------------------*)
  iterativeSolveSplitting[$u_, $residual_, hx_, hy_, AA_, Minv_, rightHandVector_, NN_,
   normf_] := Module[{residual = $residual, u = $u, sol, relativeResidual, nRow, nCol},

   {nRow, nCol} = Dimensions[u];
   residual[[2 ;; -2, 2 ;; -2]] =
    Partition[rightHandVector - AA.Flatten[u[[2 ;; -2, 2 ;; -2]]], (nCol - 2)];
   sol = Minv.rightHandVector + (Minv.NN).Flatten[u[[2 ;; -2, 2 ;; -2]]];
   u[[2 ;; -2, 2 ;; -2]] = Partition[sol, nCol - 2];

   relativeResidual = getRelativeResidual[normf, residual, hx, hy];
```

```
      {u, relativeResidual, residual}
     ];
    (*---------------------------------------------------*)
iterativeSolveGaussSeidel[$u_, $residual_, forceGrid_,
      hx_, hy_, normf_, northBCtype_, northbc_, northBCconstantValue_,
      westBCtype_, westbc_, westBCconstantValue_, eastBCtype_, eastbc_,
      eastBCconstantValue_, southBCtype_, southbc_, southBCconstantValue_, grid_] :=
     Module[{u = $u, residual = $residual, sum = 1.0 * (hx^2 + hy^2),
       product = 1.0 * hx^2 * hy^2, i, j, nRow, nCol, relativeResidual},

      {nRow, nCol} = Dimensions[u];

      For[i = 1, i ≤ nRow, i++,
       For[j = 1, j ≤ nCol, j++,
        (

         If[isEdgeCommon[i, j, nCol, nRow],
          (
           u = updateInternalEdgeNode[u, i, j, hx,
              hy, northBCtype, northBCconstantValue, northbc, southBCtype,
              southBCconstantValue, southbc, westBCtype, westBCconstantValue,
              westbc, eastBCtype, eastBCconstantValue, eastbc, grid, forceGrid]
          ),
          (

           If[isInternalNode[i, j, nCol, nRow],
            (

             residual[[i, j]] = forceGrid[[i, j]] + (1/product (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) +

                    hx^2 (u[[i - 1, j]] + u[[i + 1, j]]) - 2 * sum * u[[i, j]]));

             u[[i, j]] = 1/(2 sum) (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) + hx^2
                   (u[[i - 1, j]] + u[[i + 1, j]]) + product * forceGrid[[i, j]])
            ),
            (
             u = setCornerNodeCommon[u, i, j, nCol, nRow]
            )
           ]
          )
         ]
        )
```

```
        ]
      ];

      relativeResidual = getRelativeResidual[normf, residual, hx, hy];
      {u, relativeResidual, residual}
     ];
(*--------------------------------------------------*)
   iterativeSolveGaussSeidelRedBlack[$u_, $residual_, forceGrid_,
     hx_, hy_, normf_, northBCtype_, northbc_, northBCconstantValue_,
     westBCtype_, westbc_, westBCconstantValue_, eastBCtype_, eastbc_,
     eastBCconstantValue_, southBCtype_, southbc_, southBCconstantValue_, grid_] :=
    Module[{u = $u, residual = $residual, sum = 1.0 (hx^2 + hy^2),
      product = 1.0 * hx^2 * hy^2, i, j, nRow, nCol, k, relativeResidual},
     {nRow, nCol} = Dimensions[u];

     Do[
      For[i = 1, i ≤ nRow, i ++,
       For[j = 1, j ≤ nCol, j ++,
        If[Mod[i + j, 2] == Mod[k, 2],
         (
          If[isEdgeCommon[i, j, nCol, nRow],
           (
            u = updateInternalEdgeNode[u, i, j, hx,
               hy, northBCtype, northBCconstantValue, northbc, southBCtype,
               southBCconstantValue, southbc, westBCtype, westBCconstantValue,
               westbc, eastBCtype, eastBCconstantValue, eastbc, grid, forceGrid]
           ),
           (
            If[isInternalNode[i, j, nCol, nRow],
             (
              residual[[i, j]] = forceGrid[[i, j]] + (1/product (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) +
                     hx^2 (u[[i - 1, j]] + u[[i + 1, j]]) - 2 * sum * u[[i, j]]));
              u[[i, j]] = 1/(2 sum) (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) + hx^2
                    (u[[i - 1, j]] + u[[i + 1, j]]) + product * forceGrid[[i, j]])
             )
            ]
```

```
              )
            ]
          )]
        ]
      ],
      {k, {1, 2}}];
    relativeResidual = getRelativeResidual[normf, residual, hx, hy];
    {u, relativeResidual, residual}
   ];
(*-----------------------------------------------*)
  iterativeSolveSORPDE3[$u_, $residual_, forceGrid_, hx_, hy_, normf_,
    SORomegaUserValue_, northBCtype_, northbc_, northBCconstantValue_,
    westBCtype_, westbc_, westBCconstantValue_, eastBCtype_, eastbc_,
    eastBCconstantValue_, southBCtype_, southbc_, southBCconstantValue_, grid_] :=
   Module[{u = $u, residual = $residual, sum = 1.0 * (hx^2 + hy^2),
     product = 1.0 * hx^2 * hy^2, i, j, nRow, nCol, relativeResidual},
    {nRow, nCol} = Dimensions[u];

    For[i = 1, i ≤ nRow, i++,
     For[j = 1, j ≤ nCol, j++,
      (
       If[isEdgeCommon[i, j, nCol, nRow],
        (
         u = updateInternalEdgeNode[u, i, j, hx,
           hy, northBCtype, northBCconstantValue, northbc, southBCtype,
           southBCconstantValue, southbc, westBCtype, westBCconstantValue,
           westbc, eastBCtype, eastBCconstantValue, eastbc, grid, forceGrid]
        ),
        (
         If[isInternalNode[i, j, nCol, nRow],
          (
           residual[[i, j]] = forceGrid[[i, j]] + (1/product (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) +
                 hx^2 (u[[i - 1, j]] + u[[i + 1, j]]) - 2 * sum * u[[i, j]]));
           u[[i, j]] = SORomegaUserValue (1/(2 sum) (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) +
                  hx^2 (u[[i - 1, j]] + u[[i + 1, j]]) + product *
```

```
                 forceGrid[[i, j]]) + (1 - SORomegaUserValue) u[[i, j]]
          ),
          (
           u = setCornerNodeCommon[u, i, j, nCol, nRow]
          )
         ]
        )
       ]
      )
     ]
    ];

    relativeResidual = getRelativeResidual[normf, residual, hx, hy];
    {u, relativeResidual, residual}
   ];
(*------------------------------------------------*)
iterativeSolveSORChebyshev[$u_, $residual_, $SORChebyomega_, forceGrid_, hx_,
    hy_, normf_, stepNumber_, northBCtype_, northbc_, northBCconstantValue_,
    westBCtype_, westbc_, westBCconstantValue_, eastBCtype_, eastbc_,
    eastBCconstantValue_, southBCtype_, southbc_, southBCconstantValue_, grid_] :=
  Module[{u = $u, residual = $residual, SORChebyomega = $SORChebyomega,
     i, j, jacobiSpectralRadius, sum = 1.0 * (hx^2 + hy^2),
     product = 1.0 * hx^2 * hy^2, z, nRow, nCol, k, relativeResidual},

    {nRow, nCol} = Dimensions[u];

    z = (hx / hy) ^2;
    jacobiSpectralRadius = (Cos[Pi / (nRow - 1)] + z Cos[Pi / (nCol - 1)]) / (1. + z);

    Do[
     For[i = 1, i ≤ nRow, i++,
      For[j = 1, j ≤ nCol, j++,
       (
        If[Mod[i + j, 2] == Mod[k, 2], (*FIX ME, calculate once outside loop*)
         (
          If[isEdgeCommon[i, j, nCol, nRow],
           (
            u = updateInternalEdgeNode[u, i, j, hx,
               hy, northBCtype, northBCconstantValue, northbc, southBCtype,
```

```
                southBCconstantValue, southbc, westBCtype, westBCconstantValue,
                westbc, eastBCtype, eastBCconstantValue, eastbc, grid, forceGrid]
              ),
              (
               If[isInternalNode[i, j, nCol, nRow],
                (
                 residual[[i, j]] = forceGrid[[i, j]] + (1/product (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) +
                      hx^2 (u[[i - 1, j]] + u[[i + 1, j]]) - 2 * sum * u[[i, j]]));
                 u[[i, j]] = SORChebyomega (1/(2 sum) (hy^2 (u[[i, j - 1]] + u[[i, j + 1]]) +
                       hx^2 (u[[i - 1, j]] + u[[i + 1, j]]) + product *
                        forceGrid[[i, j]])) + (1 - SORChebyomega) u[[i, j]]
                ),
                (
                 u = setCornerNodeCommon[u, i, j, nCol, nRow]
                )
               ]
              )
             ]
            )]
          )
         ]
        ],
        {k, {1, 2}}
       ];

      If[stepNumber == 0, SORChebyomega = 1./(1 - jacobiSpectralRadius^2/2),
       SORChebyomega = 1./(1 - jacobiSpectralRadius^2/4 * SORChebyomega)];
      relativeResidual = getRelativeResidual[normf, residual, hx, hy];
      {u, relativeResidual, residual, SORChebyomega}
     ];
(*------------------------------------------------*)
```

```
solve[$u_, AA_, rightHandVector_, mask_, hx_, hy_, $relativeResidual_, $residual_,
  stationarySolver_, solverType_, splittingOrRelaxation_, $cpuTimeUsed_,
  nonStationarySolver_, Minv_, NN_, normf_, forceGrid_, SORomegaUserValue_,
  stepNumber_, $steepestDescentR_, $conjugateGradientR_, $conjugateGradientZ_,
  preconditioner_, preConditionerMatrix_, fillIn_, toleranceConstant_,
  $SORChebyomega_, $conjugateGradientP_, northBCtype_, northbc_, northBCconstantValue_,
  westBCtype_, westbc_, westBCconstantValue_, eastBCtype_, eastbc_,
  eastBCconstantValue_, southBCtype_, southbc_, southBCconstantValue_, grid_] :=
 Module[{fun, cpuTimeUsed = $cpuTimeUsed, u = $u, relativeResidual = $relativeResidual,
   residual = $residual, steepestDescentR = $steepestDescentR,
   conjugateGradientR = $conjugateGradientR, conjugateGradientZ = $conjugateGradientZ,
   conjugateGradientP = $conjugateGradientP, SORChebyomega = $SORChebyomega, cpu},

  Which[
   solverType == "direct",
   (
    {cpu, {u, relativeResidual, residual}} =
     AbsoluteTiming[directSolver[u, AA, rightHandVector, mask]];

    cpuTimeUsed = cpuTimeUsed + cpu
   ),

   solverType == "stationary" && splittingOrRelaxation == "splitting matrix method",
   (
    {cpu, {u, relativeResidual, residual}} = AbsoluteTiming[iterativeSolveSplitting[
        u, residual, hx, hy, AA, Minv, rightHandVector, NN, normf]];
    cpuTimeUsed = cpuTimeUsed + cpu
   ),

   solverType == "stationary" && splittingOrRelaxation == "relaxation method",
   (
    If[stationarySolver == "Jacobi" || stationarySolver == "Gauss-Seidel" ||
      stationarySolver == "Gauss-Seidel red/black",
     (
      fun = Which[stationarySolver == "Jacobi", iterativeSolveJacobi,
        stationarySolver == "Gauss-Seidel", iterativeSolveGaussSeidel,
        stationarySolver == "Gauss-Seidel red/black",
        iterativeSolveGaussSeidelRedBlack
       ];
      {cpu, {u, relativeResidual, residual}} = AbsoluteTiming[fun[u, residual,
         forceGrid, hx, hy, normf, northBCtype, northbc, northBCconstantValue,
         westBCtype, westbc, westBCconstantValue, eastBCtype, eastbc,
         eastBCconstantValue, southBCtype, southbc, southBCconstantValue, grid]];
      cpuTimeUsed = cpuTimeUsed + cpu
     ),
     (
      Which[stationarySolver == "SOR",
       (
        {cpu, {u, relativeResidual, residual}} =
         AbsoluteTiming[iterativeSolveSORPDE3[u, residual, forceGrid, hx, hy,
           normf, SORomegaUserValue, northBCtype, northbc, northBCconstantValue,
           westBCtype, westbc, westBCconstantValue, eastBCtype, eastbc,
```

```
                eastBCconstantValue, southBCtype, southbc, southBCconstantValue, grid]];
             cpuTimeUsed = cpuTimeUsed + cpu
            ),
            stationarySolver == "SOR/Chebyshev",
            (
             {cpu, {u, relativeResidual, residual, SORChebyomega}} = AbsoluteTiming[
               iterativeSolveSORChebyshev[u, residual, SORChebyomega, forceGrid, hx,
                hy, normf, stepNumber, northBCtype, northbc, northBCconstantValue,
                westBCtype, westbc, westBCconstantValue, eastBCtype, eastbc,
                eastBCconstantValue, southBCtype, southbc, southBCconstantValue, grid]];
             cpuTimeUsed = cpuTimeUsed + cpu
            )
           ]
          )
         ]
        ),

        solverType == "non stationary" && nonStationarySolver == "steepest descent",
        (
         {cpu, {u, relativeResidual, residual, steepestDescentR}} =
          AbsoluteTiming[steepestDescentSolve[u, residual, steepestDescentR,
            rightHandVector, hx, hy, AA, mask, stepNumber]];

         cpuTimeUsed = cpuTimeUsed + cpu
        ),
        solverType == "non stationary" && nonStationarySolver == "ConjugateGradient" &&
         (preconditioner == "SSOR" || preconditioner == "NONE"),
        (
         {cpu, {u, relativeResidual, residual, conjugateGradientR, conjugateGradientZ,
            conjugateGradientP}} = AbsoluteTiming[conjugateGradientSolve[u,
            conjugateGradientR, conjugateGradientZ, conjugateGradientP, residual,
            AA, rightHandVector, mask, stepNumber, preConditionerMatrix, hx, hy]];
         cpuTimeUsed = cpuTimeUsed + cpu
        ),
        solverType == "non stationary" && (nonStationarySolver == "ConjugateGradient" ||
           nonStationarySolver == "BiCGSTAB" || nonStationarySolver == "GMRES") &&
         (preconditioner == "ILU0" || preconditioner == "ILUT" || preconditioner == "ILUTP"),
        (
         {cpu, {u, relativeResidual, residual}} =
          AbsoluteTiming[krylovSolver[u, AA, rightHandVector, nonStationarySolver,
            preconditioner, toleranceConstant, fillIn, mask]];
         cpuTimeUsed = cpuTimeUsed + cpu
        )
       ];

       {u, relativeResidual, residual, steepestDescentR, conjugateGradientR,
        conjugateGradientZ, conjugateGradientP, SORChebyomega, cpuTimeUsed}

      ];
(*--------------------------------------------------*)
   getRelativeResidual[normf_, residual_, hx_, hy_] :=
    1.0 * If[Abs[normf] ≤ $MachineEpsilon,
```

```
      (hx * hy) ^ (1 / 4) * Norm[Flatten[residual], 2],
      ((hx * hy) ^ (1 / 4) * Norm[Flatten[residual], 2]) / normf
     ];
   (*-------------------------------------------------*)
   setOptimalSORomega[hx_, hy_, Lx_, Ly_] :=
    Module[{jacobiSpectralRadius, nIntervalsInx, nIntervalsIny, t},
     (*find optimal omega for SOR for 2D case, with non-uniform grid spacing*)
     (*see Numerical recipes, 1st edition, page 657*)
     nIntervalsInx = Lx / hx;
     nIntervalsIny = Ly / hy;

     t = (hx / hy) ^2;
     jacobiSpectralRadius = (Cos[Pi / nIntervalsInx] + t Cos[Pi / nIntervalsIny]) / (1 + t);

     2 / (1 + Sqrt[1 - jacobiSpectralRadius^2])
    ];
   (*-------------------------------------------------*)
setBoundaryConditions[$u_, grid_, northBCtype_, northbc_,
     northBCconstantValue_, westBCtype_, westbc_, westBCconstantValue_,
     eastBCtype_, eastbc_, eastBCconstantValue_, southBCtype_, southbc_,
     southBCconstantValue_] := Module[{u = $u, i, j, nRow, nCol},

     {nRow, nCol} = Dimensions[u];
     (* westbc, northbc, eastbc, and southbc, are all pure functions*)
     If[northBCtype == "Dirichlet",
      u[[1, 2 ;; nCol - 1]] =
       northBCconstantValue * Table[northbc[grid[[1, j, 1]]], {j, 2, nCol - 1}]
     ];

     If[westBCtype == "Dirichlet",
      u[[2 ;; nRow - 1, 1]] = westBCconstantValue * Table[westbc[grid[[i, 1, 2]]], {i, 2, nRow - 1}]
     ];

     If[eastBCtype == "Dirichlet",
      u[[2 ;; nRow - 1, nCol]] =
       eastBCconstantValue * Table[eastbc[grid[[i, nCol, 2]]], {i, 2, nRow - 1}]
     ];

     If[southBCtype == "Dirichlet",
      u[[nRow, 2 ;; nCol - 1]] =
       southBCconstantValue * Table[southbc[grid[[nRow, j, 1]]], {j, 2, nCol - 1}]
     ];

     (*corner points *)
     u = setCornerNodeCommon[u, 1, 1, nCol, nRow];
     u = setCornerNodeCommon[u, 1, nCol, nCol, nRow];
     u = setCornerNodeCommon[u, nRow, 1, nCol, nRow];
     u = setCornerNodeCommon[u, nRow, nCol, nCol, nRow];
     u
    ];
   (*-------------------------------------------------*)
makeSystemMatrixAndRightHandSide[u_, hx_, hy_,
```

```
northBCtype_, northbc_, northBCconstantValue_, westBCtype_, westbc_,
westBCconstantValue_, eastBCtype_, eastbc_, eastBCconstantValue_,
southBCtype_, southbc_, southBCconstantValue_, forceGrid_, grid_] := Module[
{AA, i, j, n = 0, eqs, vars, uu, U, min, An, bn, keepList, nRow, nCol, g, sum, prod},

sum = 1.0 * (hx^2 + hy^2);
prod = 1.0 * (hx^2 * hy^2);
{nRow, nCol} = Dimensions[u];
U = Array[uu[#1, #2] &, {nRow, nCol}];

With[{$nRow = nRow, $nCol = nCol},
 eqs = Table[0., {$nRow * $nCol}];
 vars = eqs
];

For[i = 1, i ≤ nRow, i++,
 For[j = 1, j ≤ nCol, j++,
   (
    n++;
    (*do corner points first *)
    If[(i == 1 && j == 1 || i == nRow && j == 1 || i == 1 && j == nCol || i == nRow && j == nCol),
      (
       {eqs, vars} = processCornersCommon[u, eqs, n, U, i, j, vars,
          northBCtype, westBCtype, southBCtype, eastBCtype, nRow, nCol]
     ), (*completed corner nodes, now check if node on edge *)
      (
       If[(i == 1 || i == nRow || j == 1 || j == nCol),
         (
          Which[i == 1,
            If[northBCtype == "Dirichlet",
              (
               eqs〚n〛 = U〚i, j〛 == u〚i, j〛;
               vars〚n〛 = U〚i, j〛
              ),
              (
               g = northBCconstantValue * northbc[grid〚i, j, 1〛];
               eqs〚n〛 = 2 * sum * U〚1, j〛 - hy^2 (U〚1, j - 1〛 + U〚1, j + 1〛) -
                   hx^2 (2 U〚2, j〛 - 2 hy * g) == prod * forceGrid〚i, j〛;
               vars〚n〛 = U〚i, j〛
              )], i == nRow,
            If[southBCtype == "Dirichlet",
              (
               eqs〚n〛 = U〚i, j〛 == u〚i, j〛;
               vars〚n〛 = U〚i, j〛
              ),
              (
               g = southBCconstantValue * southbc[grid〚i, j, 1〛];
               eqs〚n〛 = 2 sum * U〚-1, j〛 - hy^2 (U〚-1, j - 1〛 + U〚-1, j + 1〛) -
                   hx^2 (2 U〚-2, j〛 - 2 hy * g) == prod * forceGrid〚i, j〛;
               vars〚n〛 = U〚i, j〛
              )
            ], j == 1,
```

```
            If[westBCtype == "Dirichlet",
             (
              eqs[[n]] = U[[i, j]] == u[[i, j]];
              vars[[n]] = U[[i, j]]
             ),
             (
              g = westBCconstantValue * westbc[grid[[i, j, 2]]];
              eqs[[n]] = 2 sum * U[[i, 1]] - hy^2 (2 U[[i, 2]] - 2 hx g) -
                  hx^2 (U[[i - 1, 1]] + U[[i + 1, 1]]) == prod * forceGrid[[i, j]];
              vars[[n]] = U[[i, j]]
             )], j == nCol,
            If[eastBCtype == "Dirichlet",
             (
              eqs[[n]] = U[[i, j]] == u[[i, j]];
              vars[[n]] = U[[i, j]]
             ),
             (
              g = eastBCconstantValue * eastbc[grid[[i, j, 2]]];
              eqs[[n]] = 2 sum * U[[i, -1]] - hy^2 (2 U[[i, -2]] - 2 hx * g) -
                  hx^2 (U[[i - 1, -1]] + U[[i + 1, -1]]) == prod * forceGrid[[i, j]];
              vars[[n]] = U[[i, j]]
             )
            ]
           ]
          ), (*was not edge, so must be internal*)
          (
           vars[[n]] = U[[i, j]];
           eqs[[n]] = 2 sum * U[[i, j]] - hy^2 ( U[[i, j - 1]] + U[[i, j + 1]]) -
               hx^2 (U[[i - 1, j]] + U[[i + 1, j]]) == prod * forceGrid[[i, j]]
          )
         ]
        )
       ]
      )
     ]
    ];

    vars = Flatten@U;
    AA = CoefficientArrays[eqs, vars];
    keepList = obtainListOfRowsToKeep[
      nRow, nCol, northBCtype, southBCtype, westBCtype, eastBCtype];
    An = AA[[2]][[keepList, keepList]];
    bn = -AA[[1]][[keepList]];

    min = Abs[Min[An]];
    An = An * (1 / min);
    bn = bn * (1 / min);

    {An, bn}
   ];
  (*--------------------------------------------------*)
obtainListOfRowsToKeep[nRow_, nCol_, northBCtype_,
```

```
  southBCtype_, westBCtype_, eastBCtype_] := Module[{rowsToRemove = {}},

  If[northBCtype == "Dirichlet",
   AppendTo[rowsToRemove, Range[1, nCol]];
  ];
  If[southBCtype == "Dirichlet",
   AppendTo[rowsToRemove, Range[(nRow - 1) * nCol + 1, nRow * nCol]];
  ];
  If[westBCtype == "Dirichlet",
   AppendTo[rowsToRemove, Range[1, nRow * nCol, nCol]];
  ];
  If[eastBCtype == "Dirichlet",
   AppendTo[rowsToRemove, Range[nCol, nRow * nCol, nCol]];
  ];
  Complement[Range[nRow * nCol], Flatten[rowsToRemove]]
 ];
(*-----------------------------------------------*)
updateInternalEdgeNode[u_, i_, j_, hx_, hy_, northBCtype_, northBCconstantValue_,
  northbc_, southBCtype_, southBCconstantValue_, southbc_, westBCtype_,
  westBCconstantValue_, westbc_, eastBCtype_, eastBCconstantValue_, eastbc_, grid_,
  forceGrid_] := Module[{uNew = u, t1 = hx^2 + hy^2, t2 = hx^2 * hy^2, t, nRow, nCol},
  t = t2 / t1;
  {nRow, nCol} = Dimensions[u];

  If[(i == 1 || i == nRow) && (j > 1 && j < nCol), (*TOP or BOTTOM row*)
   If[i == 1, (*TOP row*)
     (
      If[northBCtype == "Neumann",
       uNew[[i, j]] = 1 / (2 t1)
           (2 u[[i + 1, j]] hx^2 - 2 hy hx^2 * northBCconstantValue * northbc[grid[[i, j, 1]]]
             + (u[[i, j - 1]] + u[[i, j + 1]]) * hx^2) - (1 / 2) t forceGrid[[i, j]]
      ]
     ),
     ((* Bottom row *)
      If[southBCtype == "Neumann",
       uNew[[i, j]] =
        1 / (2 t1)
          (2 u[[i - 1, j]] hx^2 - 2 hy hx^2 * southBCconstantValue * southbc[grid[[i, j, 1]]]
            + (u[[i, j - 1]] + u[[i, j + 1]]) * hy^2) - (1 / 2) t forceGrid[[i, j]]
      ]
     )
   ]
   ,
   (
    If[(j == 1 || j == nCol) && (i > 1 && i < nRow), (*LEFT or RIGHT edge*)
     If[j == 1,
       (
        If[westBCtype == "Neumann",
         uNew[[i, j]] =
          1 / (2 t1)
            (2 u[[i, j + 1]] hy^2 - 2 hx hy^2 * westBCconstantValue * westbc[grid[[i, j, 2]]]
              + ( u[[i - 1, 1]] + u[[i + 1, j]]) hx^2) - (1 / 2) t forceGrid[[i, j]]
```

```
          ]
         ),
         (
          If[eastBCtype == "Neumann",
           (
            uNew[[i, j]] =
             1 / (2 t1) (2 u[[i, j - 1]] hy^2 -
                 2 hx hy^2 * eastBCconstantValue * eastbc[grid[[i, j, 2]]]
                 + ( u[[i - 1, j]] + u[[i + 1, j]]) hx^2) - (1 / 2) t forceGrid[[i, j]]
           )
          ]
         )
        ]
       ]
      )
     ];

     uNew
    ];
   (*-------------------------------------------------*)
initializeSystem[nonStationarySolver_, preconditioner_, splittingOrRelaxation_,
     northBCtype_, northbc_, northBCconstantValue_, westBCtype_, westbc_,
     westBCconstantValue_, eastBCtype_, eastbc_, eastBCconstantValue_,
     southBCtype_, southbc_, southBCconstantValue_, hx_, hy_, Lx_, Ly_,
     solverType_, stationarySolver_, centerGrid_, forceTermSelection_, a_,
     b_, c_, x0_, y0_, stdx_, stdy_, initialSolution_, SORomegaUserValue_] :=
    Module[{laplacian, nRow, nCol, SORChebyomega = 1, relativeResidual = 0, grid,
      forceGrid, u, residual, residualPlotLimits, normf, mask, cpuTimeUsed = 0,
      stepNumber = 0, ok = True, statusMessage, AA, rightHandVector, factorLmatrix = {},
      factorDmatrix = {}, factorUmatrix = {}, preConditionerMatrix = {},
      Minv = {}, NN = {}, iterationMatrix = {}, nSize},

     {nRow, nCol} = getSolutionDomainDimensions[hx, hy, Lx, Ly];

     {ok, statusMessage} = isValidInput[northBCtype, westBCtype, eastBCtype,
       southBCtype, solverType, nonStationarySolver, nonStationarySolver,
       stationarySolver, preconditioner, splittingOrRelaxation];

     (*keep the grid size small for performance*)
     If[ok,
      If[Not[solverType == "direct" || solverType == "non stationary"] && nCol + nRow > 66,
       (
        statusMessage = "grid size too large for solver";
        ok = False
       )
      ]
     ];

     If[ok,
      (
       laplacian = 1 / (1 / 4)^4
         {{0, (1 / 4)^2, 0}, {(1 / 4)^2, -4 (1 / 4)^2, (1 / 4)^2}, {0, (1 / 4)^2, 0}};
```

```
(*grid contains the (x,y) physical coordinates of each grid point*)
grid = makeGridCommon[hx, hy, Lx, Ly, centerGrid];

(*evaluate the source function at each physical coordiate,
using the selected term*)
forceGrid = Which[
  forceTermSelection == 1,
  With[{$nRow = nRow, $nCol = nCol}, Table[a, {$nRow}, {$nCol}]],
  forceTermSelection == 2, Map[(#[[1]] #[[2]]) &, grid, {2}],
  forceTermSelection == 3,
  Map[(a Exp[ (#[[1]] - x0) ^2 / (2 stdx^2) + (#[[2]] - y0) ^2 / (2 stdy^2) ]) &, grid, {2}],
  forceTermSelection == 4, Map[(a (Cos[b π #[[1]]] + Sin[c π #[[2]]])) &, grid, {2}],
  forceTermSelection == 5, Map[(a (Cos[b π #[[1]]] * Sin[c π #[[2]]])) &, grid, {2}]
 ];

(*now initialize the solution grid*)
With[{$nRow = nRow, $nCol = nCol},
 u = Table[If[initialSolution == "zero", 0., RandomReal[]], {$nRow}, {$nCol}]
];

(*and initalize residual grid *)
residual = forceGrid - ListConvolve[ laplacian, u, 1];
residual[[All, 1]] = 0.0;
residual[[All, -1]] = 0.0;
residual[[1, All]] = 0.0;
residual[[-1, All]] = 0.0;

residualPlotLimits = {Min[residual], Max[residual]};

(*find the grid norm of the force grid,
use internal grid points only, since that is where*)
(*the residual is, and this value is used only for that calculation*)
normf = (hx hy) ^ (1 / 4) * Norm[Flatten[forceGrid], 2];

u = setBoundaryConditions[u, grid, northBCtype, northbc, northBCconstantValue,
  westBCtype, westbc, westBCconstantValue, eastBCtype, eastbc,
  eastBCconstantValue, southBCtype, southbc, southBCconstantValue];

mask =
 setUnknownsMask[{nRow, nCol}, northBCtype, westBCtype, eastBCtype, southBCtype];

{AA, rightHandVector} = makeSystemMatrixAndRightHandSide[
  u, hx, hy, northBCtype, northbc, northBCconstantValue, westBCtype,
  westbc, westBCconstantValue, eastBCtype, eastbc, eastBCconstantValue,
  southBCtype, southbc, southBCconstantValue, forceGrid, grid];

If[solverType == "non stationary" && (nonStationarySolver == "ConjugateGradient" ||
    nonStationarySolver == "steepest descent"),
 (
  If[Not[SymmetricMatrixQ[AA]],
   statusMessage = "incompatible solver for non-symmetric matrix";
```

```
      ok = False
     ]
    )
   ]
  )
 ];

 If[ok,
  (
   If[(solverType == "non stationary" &&
       nonStationarySolver == "ConjugateGradient" && preconditioner == "SSOR") ||
     (solverType == "stationary" && (stationarySolver == "Jacobi" ||
         stationarySolver == "Gauss-Seidel" || stationarySolver == "SOR") &&
       splittingOrRelaxation == "splitting matrix method"),
    (
     factorLmatrix = LowerTriangularize[AA, -1];
     factorDmatrix = DiagonalMatrix[Diagonal[Normal[AA]]];
     factorUmatrix = Transpose[factorLmatrix]
    )
   ];

   If[solverType == "direct" || solverType == "non stationary",
    (
     If[solverType == "non stationary" && nonStationarySolver == "ConjugateGradient",
      (
       If[preconditioner == "SSOR" || preconditioner == "NONE",
        (
         {nSize, nSize} = Dimensions[AA];
         preConditionerMatrix = generatePreConditionerMatrix[preconditioner,
           SORomegaUserValue, factorLmatrix, factorDmatrix, factorUmatrix, nSize]
        )
       ]
      )
     ]
    ),
    If[splittingOrRelaxation == "splitting matrix method" && (stationarySolver ==
         "Jacobi" || stationarySolver == "Gauss-Seidel" || stationarySolver == "SOR"),
     (
      Which[stationarySolver == "Jacobi",
       (
        Minv = Inverse[factorDmatrix];
        NN = -(factorLmatrix + factorUmatrix);
        iterationMatrix = (Minv).(NN)
       ),
       stationarySolver == "Gauss-Seidel",
       (
        Minv = Inverse[factorDmatrix + factorLmatrix];
        NN = -factorUmatrix;
        iterationMatrix = (Minv).(NN)
       ),
       stationarySolver == "SOR",
       (
```

```
            Minv = SORomegaUserValue *
              Inverse[factorDmatrix + SORomegaUserValue * factorLmatrix];
            NN = (1 / SORomegaUserValue) ((1 - SORomegaUserValue) * factorDmatrix -
                 SORomegaUserValue * factorUmatrix);
            iterationMatrix = (Minv) . (NN)
           )
          ]
         )
        ]
       ];
       statusMessage = "initialized";
      )
     ];

     {u, SORChebyomega, relativeResidual, grid, forceGrid,
      residual, residualPlotLimits, normf, mask, cpuTimeUsed, stepNumber,
      ok, statusMessage, AA, rightHandVector, factorLmatrix, factorDmatrix,
      factorUmatrix, preConditionerMatrix, Minv, NN, iterationMatrix}
    ];
   (*--------------------------------------------------*)
generatePreConditionerMatrix[preconditioner_, SORomegaUserValue_,
     factorLmatrix_, factorDmatrix_, factorUmatrix_, n_] := Module[{M},
     Which[preconditioner == "NONE",
      (
       M = IdentityMatrix[n]
      ),
      preconditioner == "SSOR",
      (
       M = 1 / (SORomegaUserValue (2 - SORomegaUserValue))
          (factorDmatrix + SORomegaUserValue factorLmatrix).Inverse[factorDmatrix].
           (factorDmatrix + SORomegaUserValue * factorUmatrix);
       M = Inverse[M]
      )
     ];
     M
    ];
   (*--------------------------------------------------*)
   makeFinalPlot[u_, grid_, Lx_, Ly_, plotToShow_, typeOfplotToShow_, plotPerformanceGoal_,
     addFaceGrids_, zAxisScale_, AA_, residual_, residualPlotLimits_] :=
    Module[{opt, tmp, cond, dim, image, n, nRow, nCol},

     {nRow, nCol} = Dimensions[u];

     opt = Which[
       typeOfplotToShow == "ListPlot3D1", {Mesh → {nRow, nCol}},
       typeOfplotToShow == "ListPlot3D2", {InterpolationOrder → 0, Filling → Bottom,
```

```
    Mesh → {nRow, nCol}, ColorFunction → "Rainbow", MeshFunctions → {#3 &}},
   typeOfplotToShow == "ListPlot3D3", {InterpolationOrder → 3,
    Mesh → {nRow, nCol}, ColorFunction → "SouthwestColors"}
  ];

image = Which[
   plotToShow == "solution",
   (
    Which[

     typeOfplotToShow == "ListPlot3D1" ||
      typeOfplotToShow == "ListPlot3D2" || typeOfplotToShow == "ListPlot3D3",
     {
      Block[{max, min},
       tmp = MapThread[Append[#1, #2] &, {grid, u}, 2];
       tmp = Chop@Flatten[tmp, 1];
       max = Max[tmp[[All, 3]]];
       min = Min[tmp[[All, 3]]];
       max = Min[max, 10^6];
       min = Max[min, -10^6];

       Item@Show[ListPlot3D[tmp,
          PerformanceGoal → plotPerformanceGoal,
          ImagePadding → {{35, 15}, {5, 1}},
          PlotRange → {Full, Full, {min, max}},
          AxesLabel →
           {Text@Style["x", Italic, 12], Text@Style["y", Italic, 12], None},
          TicksStyle → 9,
          SphericalRegion → True,
          If[addFaceGrids, FaceGrids → All, FaceGrids → None],
          If[zAxisScale == True, BoxRatios → {Lx, Ly, Min[{Lx, Ly}]}, {}],
          Sequence[opt]
         ],
         ImageSize → {ContentSizeW, ContentSizeH - 130},
         ImageMargins → 0
        ]
      ]

     },
     typeOfplotToShow == "ArrayPlot",
     {Show[ArrayPlot[u, ColorFunctionScaling -> True],
       ImagePadding → {{30, 20}, {5, 1}},
       ImageSize → {ContentSizeW, ContentSizeH - 130}
      ]
     },
     typeOfplotToShow == "ListDensityPlot",
     {
      Show[
       ListDensityPlot[Chop@Flatten[MapThread[Append[#1, #2] &, {grid, u}, 2], 1],
        PlotRange → All,
        InterpolationOrder → 0,
        ColorFunction → "SouthwestColors",
```

```
        BoundaryStyle → Black,
        ImagePadding → {{30, 20}, {5, 1}}
       ],
       ImageSize → {ContentSizeW, ContentSizeH - 130}
      ]
     },
     True,
     (
      {Show[ListContourPlot[Flatten[MapThread[Append[#1, #2] &, {grid, u}, 2], 1],
          Contours → 10,
          Evaluate[If[typeOfplotToShow == "ListContourPlot", ContourLabels →
             Function[{x, y, z}, Text[Framed[z], {x, y}]], {} ]],
          Frame → False,
          ImagePadding → {{30, 20}, {5, 1}}
         ], ImageSize → {ContentSizeW, ContentSizeH - 130}]}
     )
    ]
   ),
   plotToShow == "solution data",
   (
    {makeScrolledPaneCommon[Normal@u, ContentSizeH - 10, ContentSizeW]}
   ),
   plotToShow == "system matrix information",
   (
    cond = LUDecomposition[AA][[3]];
    dim = Dimensions[AA];
    n = Min[30, First@dim];

    {Grid[{
       {Style[Text@Row[{"condition number = ", cond}], 12]},
       {Style[Text@Row[{"matrix size = ", dim}], 12]},
       {Style[Text["eigenvalues"], 12]},
       {makeScrolledPaneCommon[
         Transpose@Partition[Eigenvalues[Normal@AA, n], 1], 45, ContentSizeW - 20]},
       {Style[Text["A matrix"], 12]},
       {makeScrolledPaneCommon[
         Normal@AA⟦1 ;; n, 1 ;; n⟧, ContentSizeH - 140, ContentSizeW]}
      }]
    }
   ),
   plotToShow == "residual",
   (
    Which[
     typeOfplotToShow == "ListPlot3D1" ||
      typeOfplotToShow == "ListPlot3D2" || typeOfplotToShow == "ListPlot3D3",
     {
      tmp = MapThread[Append[#1, #2] &, {grid, residual}, 2];

      (*watch out for PlotRange→All and conflict with BoxRatios*)
      Item@Show[ListPlot3D[Chop@Flatten[tmp, 1],
         PerformanceGoal → plotPerformanceGoal,
         ImagePadding → {{30, 20}, {5, 1}},
```

```
            If[zAxisScale == True,
             PlotRange → Automatic, PlotRange → {All, All, residualPlotLimits}],
            AxesLabel → {Text@Style["x", 12], Text@Style["y", 12], None},
            PlotLabel → Text@Style["residual", 12],
            TicksStyle → 9,
            SphericalRegion → True,
            If[addFaceGrids, FaceGrids → All, FaceGrids → None],
            If[zAxisScale == True, BoxRatios → {Lx, Ly, Min[{Lx, Ly}]}, {}],
            AspectRatio → 1.2,
            Sequence[opt]
           ], ImageSize → {ContentSizeW, ContentSizeH - 130}]},

        typeOfplotToShow == "ListDensityPlot",
        {
         Show[
          ListDensityPlot[
           Chop@Flatten[MapThread[Append[#1, #2] &, {grid, residual}, 2], 1],
           PlotRange → All,
           Mesh → None,
           InterpolationOrder → 0,
           ColorFunction → "SouthwestColors",
           BoundaryStyle → Black,
           ImagePadding → {{30, 20}, {5, 1}}
          ],
          ImageSize → {ContentSizeW, ContentSizeH - 130}
         ]
        },
        typeOfplotToShow == "ArrayPlot",
        {Show[ArrayPlot[residual, ColorFunctionScaling -> True],
          ImagePadding → {{30, 20}, {5, 1}},
          ImageSize → {ContentSizeW, ContentSizeH - 130}]
        },

        True,
        {Show[
          ListContourPlot[Flatten[MapThread[Append[#1, #2] &, {grid, residual}, 2], 1],
           Contours → 10,
           Evaluate[If[typeOfplotToShow == "ListContourPlot",
             ContourLabels → Function[{x, y, z}, Text[Framed[z], {x, y}]], {} ]],
           Frame → False
          ], ImageSize → {ContentSizeW, ContentSizeH - 130}]}
       ]
      )
     ];

    image
   ];
  (*--------------------------------------------------*)
isValidInput[northBCtype_, westBCtype_, eastBCtype_, southBCtype_,
    solverType_, nonStationarySolver_, nonStationarySolver_, stationarySolver_,
    preconditioner_, splittingOrRelaxation_] := Module[{statusMessage = ""},
```

```
    If[northBCtype == "Neumann" &&
      westBCtype == "Neumann" && eastBCtype == "Neumann" && southBCtype == "Neumann",
     statusMessage = "detected all boundaries with Neumann B.C.";

     Return[{False, statusMessage}]
    ];

    If[solverType == "non stationary" &&
      (nonStationarySolver == "BiCGSTAB" || nonStationarySolver == "GMRES") &&
      (preconditioner == "SSOR" || preconditioner == "NONE"),
     statusMessage = "solver supports only ILU0, ILUP or ILUTP";
     Return[{False, statusMessage}]
    ];

    If[solverType == "non stationary" &&
      nonStationarySolver == "steepest descent" && Not[preconditioner == "NONE"],
     statusMessage = "solver accepts only NONE preconditioner";
     Return[{False, statusMessage}]
    ];

    If[solverType == "stationary" && splittingOrRelaxation == "splitting matrix method" &&
      (stationarySolver == "Gauss-Seidel red/black" ||
        stationarySolver == "SOR/Chebyshev"),
     statusMessage = "solver does not support splitting method";
     Return[{False, statusMessage}]
    ];

    If[solverType == "stationary" &&
      splittingOrRelaxation == "splitting matrix method" && (northBCtype == "Neumann" ||
        westBCtype == "Neumann" || eastBCtype == "Neumann" || southBCtype == "Neumann"),
     statusMessage = "splitting method does not support Neumann";
     Return[{False, statusMessage}]
    ];

    {True, statusMessage}
   ];
  (*---------------------------------------*)
  (* Thanks to Heike @SO for this function *)
  (*---------------------------------------*)
myGrid[tab_, opts___] := Module[{divlocal, divglobal, pos},
    (*extract option value of Dividers from opts to divglobal*)
    (*default value is {False,False}*)
    divglobal = (Dividers /. {opts}) /. Dividers → {False, False};
    (*transform divglobal so that it is in the form {colspecs,rowspecs}*)
    If[Head[divglobal] =!= List, divglobal = {divglobal, divglobal}];
    If[Length[divglobal] == 1, AppendTo[divglobal, False]];
    (*Extract positions of dividers between rows from tab*)
    pos = Position[tab, Dividers → _, 1];
    (*Build list of rules for divider specifications between rows*)
    divlocal = MapIndexed[# - #2[[1]] + 1 → Dividers /. tab[[#]] &, Flatten[pos]];
    (*Final settings for dividers are {colspecs,{rowspecs,divlocal}}*)
    divglobal[[2]] = {divglobal[[2]], divlocal};
```

```
     Grid[Delete[tab, pos], Dividers → divglobal, opts]
    ];
(*-----------------------------------------------*)
   MakeBoxes[Derivative[indices__][f_][vars__], TraditionalForm] :=
    SubscriptBox[MakeBoxes[f, TraditionalForm], RowBox[Map[ToString, Flatten[
        Thread[dummyhead[{vars}, Partition[{indices}, 1]]] /. dummyhead → Table]]]];
   (*-----------------------------------------------*)
   ContentSizeW = 230;
   ContentSizeH = 405 ;

  }
]
```