
EPToolbox package file

This notebook auto-generates the `EPToolbox.m` package file. For examples and documentation see the `EPToolbox usage.nb` notebook.

© Emilio Pisanty, 2014-2016, available under the CC-BY-SA 4.0 license.

Package Start

```
BeginPackage["EPToolbox`"];
```

Complex root finder

```
FindComplexRoots::usage =  
  "FindComplexRoots[e1==e2, {z, zmin, zmax}] attempts to find complex roots of  
  the equation e1==e2 in the complex rectangle with corners zmin and zmax.  
  
FindComplexRoots[{e1==e2, e3==e4, ...}, {z1, z1min, z1max},  
  {z2, z2min, z2max}, ...] attempts to find complex roots of the  
  given system of equations in the multidimensional complex  
  rectangle with corners z1min, z1max, z2min, z2max, ...";  
Seeds::usage = "Seeds is an option for FindComplexRoots which determines how many  
  initial seeds are used to attempt to find roots of the given equation.";  
SeedGenerator::usage = "SeedGenerator is an option for FindComplexRoots  
  which determines the function used to generate the seeds for  
  the internal FindRoot call. Its value can be RandomComplex,  
  RandomNiederreiterComplexes, RandomSobolComplexes,  
  DeterministicComplexGrid, or any function f such that f[{zmin, zmax}, n]  
  returns n complex numbers in the rectangle with corners zmin and zmax."  
  
Options[FindComplexRoots] = Join[Options[FindRoot], {Seeds -> 50,  
  SeedGenerator -> RandomComplex, Tolerance -> Automatic, Verbose -> False}];  
SyntaxInformation[FindComplexRoots] = {"ArgumentsPattern" ->  
  {_, {_, _, _}, OptionsPattern[]}, "LocalVariables" -> {"Table", {2, ∞}}};  
FindComplexRoots::seeds =  
  "Value of option Seeds -> `1` is not a positive integer.";  
FindComplexRoots::tol =  
  "Value of option Tolerance -> `1` is not Automatic or a number in [0,∞).";  
$MessageGroups = Join[$MessageGroups, {"FindComplexRoots" -> {FindRoot::lstol}}]  
  
Protect[Seeds];  
Protect[SeedGenerator];  
  
Begin["`Private`"];  
FindComplexRoots[equations_List, domainSpecifiers__, ops : OptionsPattern[]] :=  
  Block[{seeds, tolerances},  
    If[! IntegerQ[Rationalize[OptionValue[Seeds]]] || OptionValue[Seeds] ≤ 0,
```

```

Message[FindComplexRoots::seeds, OptionValue[Seeds]]];
If[! (OptionValue[Tolerance] === Automatic || OptionValue[Tolerance] ≥ 0),
  Message[FindComplexRoots::tol, OptionValue[Seeds]]];

seeds = OptionValue[SeedGenerator][
  {domainSpecifiers}[[All, {2, 3}]], OptionValue[Seeds]];
tolerances = Which[
  ListQ[OptionValue[Tolerance]], OptionValue[Tolerance],
  True, ConstantArray[
    Which[
      NumberQ[OptionValue[Tolerance]], OptionValue[Tolerance],
      True, 10^If[NumberQ[OptionValue[WorkingPrecision]],
        2 - OptionValue[WorkingPrecision], 2 - $MachinePrecision]
    ]
  , Length[{domainSpecifiers}]]
];

If[OptionValue[Verbose], Hold[], Hold[FindRoot::1stol]] /. {
  Hold[messageSequence___] :> Quiet[
    DeleteDuplicates[
      Select[
        Check[
          FindRoot[
            equations
            , Evaluate[Sequence @@ Table[{domainSpecifiers}[[j, 1]],
              #[[j]], {j, Length[{domainSpecifiers}]}]]
            , Evaluate[Sequence @@ FilterRules[{ops}, Options[FindRoot]]]]
        ],
        ### &[]
      ] & /@ seeds,
    Function[
      repList,
      ReplaceAll[
        Evaluate[And @@ Table[
          And[
            Re[{domainSpecifiers}[[j, 2]]] ≤ Re[
              {domainSpecifiers}[[j, 1]] ≤ Re[{domainSpecifiers}[[j, 3]]],
            Im[{domainSpecifiers}[[j, 2]]] ≤ Im[{domainSpecifiers}[[
              j, 1]]] ≤ Im[{domainSpecifiers}[[j, 3]]]
          ]
        , {j, Length[{domainSpecifiers}]}]]]
      , repList]
    ]
  ],
  Function[{repList1, repList2},
    And @@ Table[
      Abs[{domainSpecifiers}[[j, 1]] /. repList1] -

```

```

        ({domainSpecifiers}[[j, 1]] /. repList2)] < tolerances[[j]]
      , {j, Length[{domainSpecifiers}]]}
    ]
  ], {messageSequence}]]}
]
FindComplexRoots[e1_ == e2_, {z_, zmin_, zmax_}, ops : OptionsPattern[]] :=
  FindComplexRoots[{e1 == e2}, {z, zmin, zmax}, ops]
End[];

```

Quasirandom number generators

RandomSobolComplexes

```

RandomSobolComplexes::usage =
  "RandomSobolComplexes[{zmin, zmax}, n] generates a low-discrepancy
  Sobol sequence of n quasirandom complex numbers
  in the rectangle with corners zmin and zmax.

RandomSobolComplexes[{{z1min,z1max},{z2min,z2max},...},n] generates a
  low-discrepancy Sobol sequence of n quasirandom complex numbers in the
  multi-dimensional rectangle with corners {z1min,z1max},{z2min,z2max},....";

Begin["`Private`"];
RandomSobolComplexes[pairsList__, number_] := Map[
  Function[randomsList,
    pairsList[[All, 1]] + Complex@@@Times[
      ReIm[pairsList[[All, 2]] - pairsList[[All, 1]],
      randomsList
    ]
  ],
  BlockRandom[
    SeedRandom[
      Method -> {"MKL", Method -> {"Sobol", "Dimension" -> 2 Length[pairsList]}}];
    SeedRandom[];
    RandomReal[{0, 1}, {number, Length[pairsList], 2}]
  ]
]
RandomSobolComplexes[{zmin_?NumericQ, zmax_?NumericQ}, number_] :=
  RandomSobolComplexes[{{zmin, zmax}}, number][[All, 1]]
End[];

```

RandomNiederreiterComplexes

```
RandomNiederreiterComplexes::usage =
  "RandomNiederreiterComplexes[{zmin, zmax}, n] generates
   a low-discrepancy Niederreiter sequence of n quasirandom
   complex numbers in the rectangle with corners zmin and zmax.

RandomNiederreiterComplexes[{{z1min,z1max},{z2min,z2max},...},n]
  generates a low-discrepancy Niederreiter sequence of
  n quasirandom complex numbers in the multi-dimensional
  rectangle with corners {z1min,z1max},{z2min,z2max},....";

Begin["`Private`"];
RandomNiederreiterComplexes[pairsList__, number_] := Map[
  Function[randomsList,
    pairsList[[All, 1]] + Complex@@@Times[
      ReIm[pairsList[[All, 2]] - pairsList[[All, 1]],
      randomsList
    ]
  ],
  BlockRandom[
    SeedRandom[Method →
      {"MKL", Method → {"Niederreiter", "Dimension" → 2 Length[pairsList]}}];
    SeedRandom[];
    RandomReal[{0, 1}, {number, Length[pairsList], 2}]
  ]
];
RandomNiederreiterComplexes[{zmin_?NumericQ, zmax_?NumericQ}, number_] :=
  RandomNiederreiterComplexes[{{zmin, zmax}}, number][[All, 1]]
End[];
```

DeterministicComplexGrid

```
DeterministicComplexGrid::usage =
  "DeterministicComplexGrid[{zmin, zmax}, n] generates a grid of about n equally
   spaced complex numbers in the rectangle with corners zmin and zmax.

DeterministicComplexGrid[{{z1min,z1max},{z2min,z2max},...},n] generates
  a regular grid of about n equally spaced complex numbers in the
  multi-dimensional rectangle with corners {z1min,z1max},{z2min,z2max},....";
```

```

Begin["`Private`"];
DeterministicComplexGrid[pairsList_, number_] :=
Block[{sep, separationsList, gridPointBasis, k},
  sep = NestWhile[0.99 # &,
    Min[Flatten[ReIm[pairsList[[All, 2]] - pairsList[[All, 1]]]], Times @@  $\frac{1}{0.99 \#}$  Floor[
      Flatten[ReIm[pairsList[[All, 2]] - pairsList[[All, 1]]], 0.99 #] ≤ number &];
  separationsList = Round[ $\frac{1}{sep}$  Floor[Flatten[
    ReIm[pairsList[[All, 2]] - pairsList[[All, 1]]], sep]]];
  gridPointBasis = MapThread[
    Function[{1, n}, Range[1[[1]], 1[[2]],  $\frac{1[[2]] - 1[[1]]}{n + 1}$ ] [[2 ;; -2]]],
    {Flatten[Transpose[ReIm[pairsList], {1, 3, 2}], 1], separationsList}
  ];
  Flatten[Table[
    Table[k[2 j - 1] + i k[2 j], {j, 1, Length[pairsList]}],
    Evaluate[
      Sequence@@Table[{k[j], gridPointBasis[[j]]}, {j, 1, 2 Length[pairsList]}]]
  ], Evaluate[Range[1, 2 Length[pairsList]]]]
]
DeterministicComplexGrid[{zmin_?NumericQ, zmax_?NumericQ}, number_] :=
  DeterministicComplexGrid[{zmin, zmax}, number] [[All, 1]]
End[];

```

RandomComplex

Updating RandomComplex to handle input of the form RandomComplex[{0, 1+i}, {2, 3+i}, n].

```

Begin["`Private`"];
Unprotect[RandomComplex];
RandomComplex[{range1_List, moreRanges___}, number_] :=
  Transpose[RandomComplex[#, number] & /@ {range1, moreRanges}]
Protect[RandomComplex];
End[];

```

Contour plot cleaner

This function cleans up automatically generated contour plots. Generically, a contour plot is made of a Polygon with a vast number of vertices in its interior, which are not necessary and only slow the plot down - including a large use of CPU when the mouse hovers above it, which is definitely unwanted. (In addition, these polygons can give rise to white edges inside each contour when printed to pdf, which is also undesirable.) This function changes such Polygons to FilledCurve constructs which no longer contain the unwanted mid-contour points.

This function was written by Szabolcs Horvát

(<http://mathematica.stackexchange.com/users/12/szabolcs>) and was originally posted at <http://mathematica.stackexchange.com/a/3279> under a CC-BY-SA license.

```
cleanContourPlot::usage =
  "cleanContourPlot[plot] Cleans up a contour plot by coalescing
    complex polygons into single FilledCurve instances.
    See MM.SE/a/3279 for source and documentation.";

Begin["`Private`"];
cleanContourPlot[cp_] :=
Module[{points, groups, regions, lines},
  groups =
    Cases[cp, {style_, g_GraphicsGroup} :> {{style}, g}, Infinity];
  points =
    First@Cases[cp, GraphicsComplex[pts_, ___] :> pts, Infinity];
  regions = Table[
    Module[{group, style, polys, edges, cover, graph},
      {style, group} = g;
      polys = Join @@ Cases[group, Polygon[pt_, ___] :> pt, Infinity];
      edges = Join @@ (Partition[#, 2, 1, 1] & /@ polys);
      cover = Cases[Tally[Sort /@ edges], {e_, 1} :> e];
      graph = Graph[UndirectedEdge @@@ cover];
      {Sequence @@ style,
       FilledCurve[
        List /@ Line /@ First /@
          Map[First,
            FindEulerianCycle /@ (Subgraph[graph, #] &) /@
              ConnectedComponents[graph], {3}]}]}
  ],
  {g, groups}];
  lines = Cases[cp, _Tooltip, Infinity];
  Graphics[GraphicsComplex[points, {regions, lines}],
    Sequence @@ Options[cp]]
]
End[];
```

Dynamics profiler

This function produces a profiling suite for any dynamics constructs, which can be used to see which parts of a Dynamic application take up the most processing time and calls.

This function was written by Rui Rojo (<http://mathematica.stackexchange.com/users/109/rojo>) and was originally posted at <http://mathematica.stackexchange.com/a/8047> under a CC-BY-SA license.

```
profileDynamics::usage =
  "profileDynamics[dynamicsConstruct] Produces a profiling suite
    for the Dynamic statements in its argument.
    See MM.SE/a/8047 for source and documentation.";
```

```

Begin["`Private`"];
ClearAll[profileDynamics];
Options[profileDynamics] = {"Print" -> False};
profileDynamics[d_, OptionsPattern[]] := With[
  {print = OptionValue["Print"]},
  Module[{counter = {}},
    DynamicModule[
      {diag, start, tag},
      diag[] := CreateDocument[Column[{
        Button["Reset counter", counter = start],
        Dynamic@Grid[Join[
          {"Dynamic expression", "Count", "Time"}],
          MapAt[Short, #, 1] & /@ counter
        ]]]];
      CellPrint@
      ExpressionCell[Button["See profiling information", diag[]]];
      d //. {
        i : Annotation[_ , {tag, ___}] :> i,
        e : Dynamic[sth : Except[First[{_, tag}]], rest___] :> With[
          {pos = 1 + Length@counter,
           catalog =
            Annotation[
              InputForm@e, {tag, Unique["profileDynamics`annot"]}]],
          AppendTo[counter, {catalog, 0, 0.}];
          Dynamic[First@{Refresh[
            If[print, Print[catalog]]; ++counter[[pos, 2]];
            (counter[[pos, 3]] += First@#; Last@#) &[
              AbsoluteTiming[Refresh[sth]]],
            None], tag}, rest] /; True
          ]
        } // (start = counter; #) &
      ]
    ]
  ]
End[];

```

Package End

```
EndPackage[];
```