23. Procedures and Packages

2015-06-15

```
$Version
```

```
10.0 for Mac OS X x86 (64-bit) (September 10, 2014)
```

23.1 Setting up *Mathematica* Packages

In a typical *Mathematica* package there are generally two kinds of new symbols. Those of the first kind are destined for "export", i.e. for use outsidethe package. The second kind are those used within the package. It is useful to put these two kinds of symbols into different contexts.

The usual convention is to put symbols for export in a context with a name **`Package``** that corresponds to the name of the package. Whenever the package is read in, it adds this context to the context search path, so that the symbols in this context can be refered to by their short names.

Symbols that are not intended for export, but are ones used within the package, are conventionally put into a context with the name **`Package`Private``.** This context is not added to the context search path. As a result, the symbols in this context cannot be accessed except by giving their full names.

`Package``	symbols for export
`Package`Private`	symbols for internal use only
`System``	built-in *Mathematica* symbols
`Needed1`,Needed2`,...`	other contexts needed in the package

There is a standard sequence of *Mathematica* commands to set up contexts in a package. These set the values of **`$Context`** and **`$ContextPath`** so that the new symbols which are introduced are created in the appropriate contexts.

`BeginPackage["Package`"]`	set **`Package``** to be the current context, and put only **`System``** on the context search path
`f::usage = "text",...`	introduce the objects intended for export
`Begin["`Private`"]`	set the current context to **`Package`Private``**
`f[args] = value, ...`	give main body of definitions in the pack.
`End[]`	revert to previous context (here **`Package``**)
`EndPackage[]`	end the package, prepending the **`Package``** to the context search path.

```
BeginPackage["SinSer`"]
SinSer::usage = " Series expansion for Sin[x]."
Begin["Private`"]
SinSer[x_, n_] :=
Module[{i}, Print[Sum[ I (-I x)^i /(2 i - 1)!, {i, 1, n, 2} ]]
]
End[]
EndPackage[]
```

```
SinSer`

 Series expansion for Sin[x].

Private`

Private`
```

```
SinSer[x,5]
```

$$x - \frac{x^3}{120} + \frac{x^5}{362\,880}$$

```
SinSer[0.1, 5]

0.0999917 + 0. i

Sin[0.1]

0.0998334
```

```
BeginPackage["SinSer`"]
SinSerr::usage = "Series expansion for Sin[x]   ."
Begin["Private`"]
SinSerr[x_, n_] :=
Module[{i, a[1] = 1, a[3] = -1/6, a[5] = 1/120},
        Sum[ a[i] x^i, {i, 1, n, 2} ]
]
End[]
EndPackage[]
SinSer`

Series expansion for Sin[x]   .

Private`

Private`
```

```
SinSerr[x,5]
```

Module::lvset :
  Local variable specification {Private`i$, Private`a[1] = 1, Private`a[3] = $-\frac{1}{6}$, Private`a[5] = $\frac{1}{120}$} contains Private`a[1] = 1,
    which is an assignment to Private`a[1]; only assignments to symbols are allowed. >>

Module[{Private`i$, Private`a[1] = 1, Private`a[3] = $-\frac{1}{6}$, Private`a[5] = $\frac{1}{120}$},
 Sum[Private`a[Private`i$] $x^{\text{Private`i\$}}$, {Private`i$, 1, 5, 2}]]

```
Module::"lvset" :
 "Local variable specification {Private`i$, Private`a[1] = 1,Private`a[3] = 1/6,
   Private`a[5] = 1/120}, contains Private`a[1] = 1, which is an
   assignment to Private`a[1]; only assignments to symbols are allowed. >>
```

23.2 Collecting Commands in Package

In this section it is shown by way of an example how a package is assembled; this is done in several steps in which the number of procedures is incrementedor particular procedures are refined. This exemplary package contains several procedures which map parts of plane on an other plane. This is done by complex mappings. Except for some examples, the material is taken from R. Maeder: Programming in *Mathematica*.

23.2.1 Plotting the Coordinate Lines in the Complex Plane

At first a rectangular grid is generated by connecting lattice points by straight lines.

```
points = Table[N[x + I y], {x, -Pi/2, Pi/2, Pi/14}, {y, -1, 1, 2/10}];
Short[points, 3]

{{-1.5708 - 1. i, -1.5708 - 0.8 i, -1.5708 - 0.6 i,
   <<6>>, -1.5708 + 0.8 i, -1.5708 + 1. i}, <<13>>, {<<1>>, <<10>>}}

coords = Map[ {Re[#], Im[#]}&, points, {2} ];
Short[coords, 3]

{{{-1.5708, -1.}, {-1.5708, -0.8}, {-1.5708, -0.6},
   <<6>>, {-1.5708, 0.8}, {-1.5708, 1.}}, <<13>>, {<<1>>}}
```

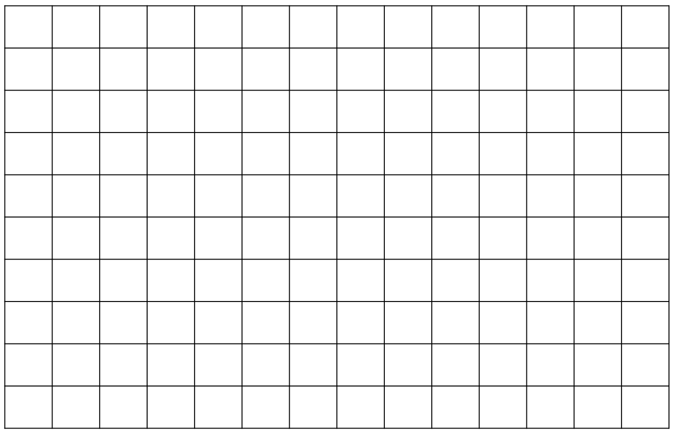
```
vlines = Map[ Line, coords];
```

```
Short[ FullForm[vlines] ,13]
```

```
List[Line[List[List[-1.5707963267948966`, -1.`],
   List[-1.5707963267948966`, -0.8`], List[-1.5707963267948966`, -0.6`],
   List[-1.5707963267948966`, -0.4`], List[-1.5707963267948966`, -0.2`],
   List[-1.5707963267948966`, 0], List[-1.5707963267948966`, 0.2`],
   List[-1.5707963267948966`, 0.4`], List[-1.5707963267948966`, 0.6`],
   List[-1.5707963267948966`, 0.8`], List[-1.5707963267948966`, 1.`]]],
 Line[\[LeftSkeleton]1\[RightSkeleton]],
 Line[List[List[-1.121997376282069`, -1.`],
   List[-1.121997376282069`, -0.8`], List[-1.121997376282069`, -0.6`],
   List[-1.121997376282069`, -0.4`], \[LeftSkeleton]3\[RightSkeleton],
   List[-1.121997376282069`, 0.4`], List[-1.121997376282069`, 0.6`],
   List[-1.121997376282069`, 0.8`], List[-1.121997376282069`, 1.`]]],
 Line[\[LeftSkeleton]1\[RightSkeleton]],
 \[LeftSkeleton]7\[RightSkeleton],
 \[LeftSkeleton]1\[RightSkeleton],
 Line[List[\[LeftSkeleton]1\[RightSkeleton]]],
 Line[\[LeftSkeleton]1\[RightSkeleton]],
 Line[\[LeftSkeleton]1\[RightSkeleton]]]
```

```
hlines = Map[Line, Transpose[coords]];
```

```
Show[Graphics[Join[vlines, hlines]]]
```



The complex map

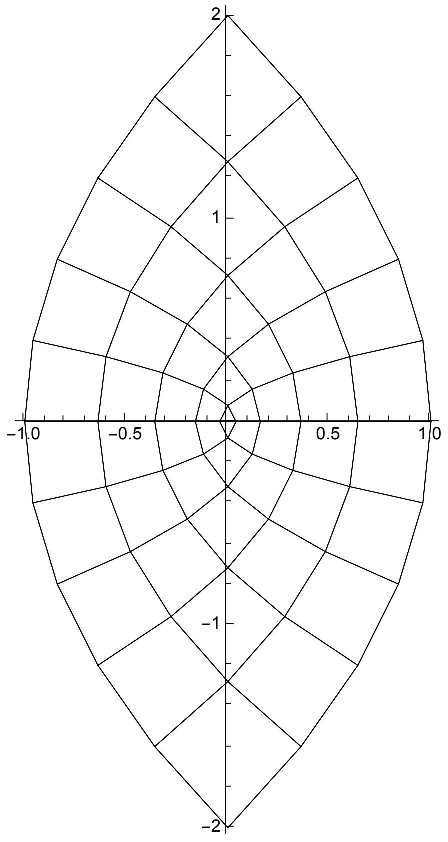
$$w = u + i\, v = f(z) = f(x + i\, y)$$

maps the above rectangular grid into a new orthogonal grid; in most cases this is curvlinear. In particular, the function
$w = z^2$ gives two sets of hyperbolas:

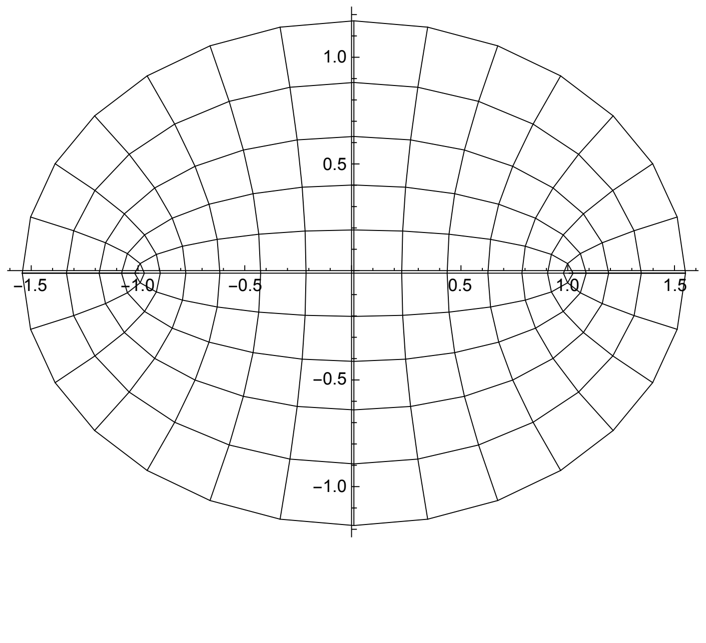
$$u = x^2 - y^2 = c1 \quad \text{and} \quad v = 2\, x\, y = c2\,.$$

```
points = Table[N[(x + I y)^2],  {x, -1, 1, 2/10},
                                {y, -1, 1, 2/10}];
coords = Map[ {Re[#], Im[#]}&, points, {2} ];
lines = Map[ Line, Join[coords, Transpose[coords]] ];
```

```
pv = Show[Graphics[lines], Axes → Automatic]
```



```
points = Table[N[Sin[x + ⅈ y]], {x, -π/2, π/2, π/14}, {y, -1, 1, 2/10}];
coords = Map[{Re[#1], Im[#1]} &, points, {2}];
lines = Line /@ Join[coords, Transpose[coords]];
Show[Graphics[lines], Axes → Automatic]
```



23.2.2 Putting the Commands into a Function

```
CartesianMap[ func_, {x0_, x1_, dx_}, {y0_, y1_, dy_} ] :=
 Module[ {x, y, coords, ulines, vlines},
   coords = Table[ N[func[x + I y]], {x, x0, x1, dx},
                  {y, y0, y1, dy} ];
   coords = Map[ {Re[#], Im[#]} &, coords, {2} ];
   ulines = Map[ Line, coords ];
   vlines = Map[ Line, Transpose[coords] ];
   Show[ Graphics[Join[ulines, vlines],
     AspectRatio -> Automatic, Axes -> Automatic ] ]
 ]
```

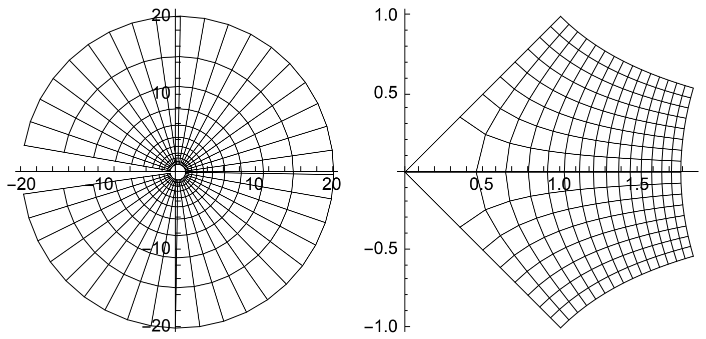
All variables local to `CartesianMap[]` are declared in the `Module[]` statement. It is important that the variables **x** and **y** do not have values inside `CartesianMap` as they are used as variables in an iterator.

The first argument of `CartesianMap` is the name of the function to be plotted. It is later used as the head of an expresssion in **func[x + I y].** It we want to specify a function that is not built in we can either write a definition for it beforehand or use a pure function as the argument.

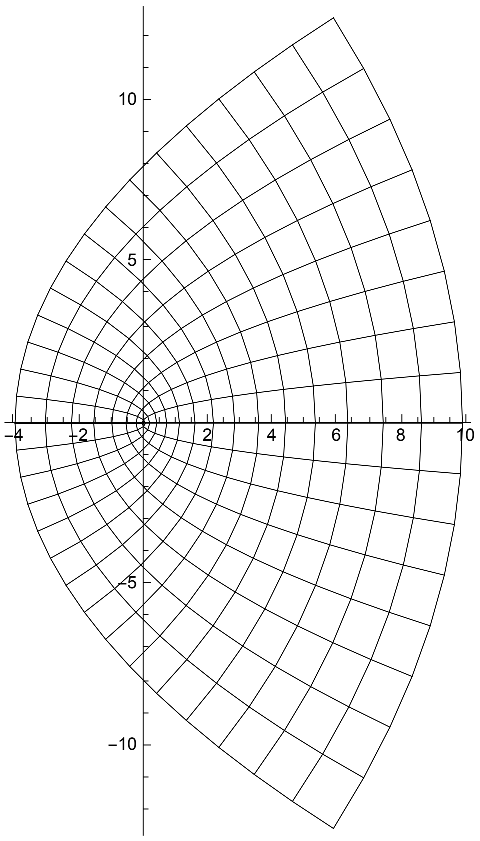
```
ve = CartesianMap[Exp, {0, 3, .3}, {-3, 3, Pi/20}];
```

```
vs = CartesianMap[ Sqrt, {0, Pi, Pi/15}, {-2, 2, 4/16} ];
```

```
Show[GraphicsRow[{ve, vs}]]
```



```
CartesianMap[ #^2 &, {0, Pi, Pi/15}, {-2,2,4/16} ]
```



The last command gives the same figure (intersecting hyperbolas) as obtained in the preceeding subsection. $\mathbf{w = z^2}$.

23.2.3 The Basis Ingredients for a Package

For a procedure or a package of procedures to be portable, it is important that the symbols (variables and functions) internal to the procedure are local, i.e. they must be separated from the symbols used by the programme calling the procedures. This is achieved with the help of the commands `Module` or `Block` (cf. Chap. 22). These put the symbols into a separate context. This context, however, must be visible, so that we can use the functions later on. This is achieved by the pair of commands: `BeginPackage[]` and `EndPackage[].`

22.2.4 A Package Context for CartesianMap

```
BeginPackage["MyCartesianMap`"]

MyCartesianMap::usage="MyCartesianMap[f, {x0,x1,dx}, {y0,y1,dy}]
   plots the image of the cartesian coordinate lines under
   the function  f."

Begin["`Private`"]

MyCartesianMap[ func_, {x0_, x1_, dx_}, {y0_, y1_, dy_} ] :=
   Module[ {x, y, coords, lines},
      coords = Table[ N[func[x + I y]], {x, x0, x1, dx},
                                        {y, y0, y1, dy}  ];
      coords = Map[ {Re[#], Im[#]}&, coords, {2} ];
      lines  = Map[ Line, Join[coords, Transpose[coords]] ];
      Show[ Graphics[lines], AspectRatio -> Automatic,
                             Axes        -> Automatic  ]
]
End[]
EndPackage[]
```

```
MyCartesianMap`

MyCartesianMap[f, {x0,x1,dx}, {y0,y1,dy}]
   plots the image of the cartesian coordinate lines under
   the function  f.

MyCartesianMap`Private`

MyCartesianMap`Private`
```

The initial ` in the context name inside the comand **Begin["`Private`"]** establishes **`Private`** as a subcontext of the **Context CartesianMap`** (so its full name is **CartesianMap`Private`).**

```
Context[MyCartesianMap]
```

```
MyCartesianMap`
```

```
$ContextPath
```

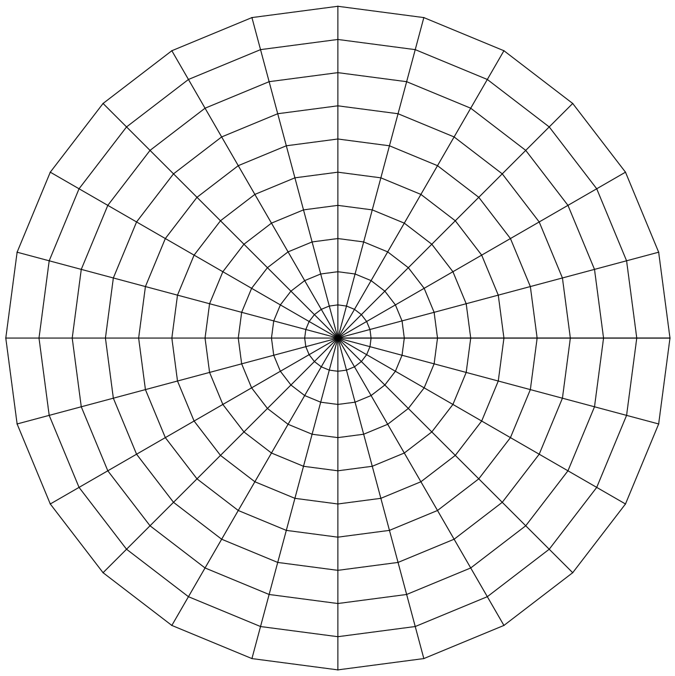
```
{MyCartesianMap`, SinSer`, TemplatingLoader`, PacletManager`, System`, Global`}
```

23.2.5 Adding another Function to the Package

Now a second function, **MyPolarMap[],** is added.

$$z = r\,e^{i\varphi}$$

```
points = Table[N[r Exp[ⅈ phi]], {r, 0, 1, 0.1`}, {phi, 0, 2 π, 2 π/24}];
coords = Map[{Re[#1], Im[#1]} &, points, {2}];
vlines = Line /@ coords; hlines = Line /@ Transpose[coords];
Show[Graphics[Join[hlines, vlines]], AspectRatio → Automatic]
```



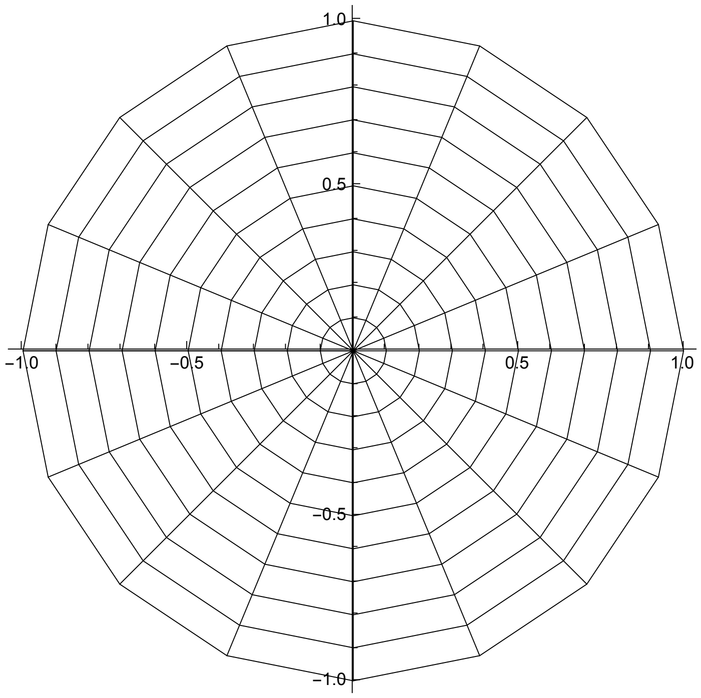
Since all the calculations are the same except that the complex numbers are generated as **`r Exp[I φ]`** in place of **`(x + I y)`**, the same code can be used which was used for the function **`CartesianMap[]`**. This common code is put into a separate auxiliary function **`MakeLines[]`** that is then used inside **`CartesianMap[]`** and **`PolarMap[]`**. This function should take a matrix of complex numbers as input and return a **`Gaphics[]`** object that will plot the lines corresponding to the rows and columns of the matrix.

Both functions are in the same package now called **`MyComplexMap.m`**

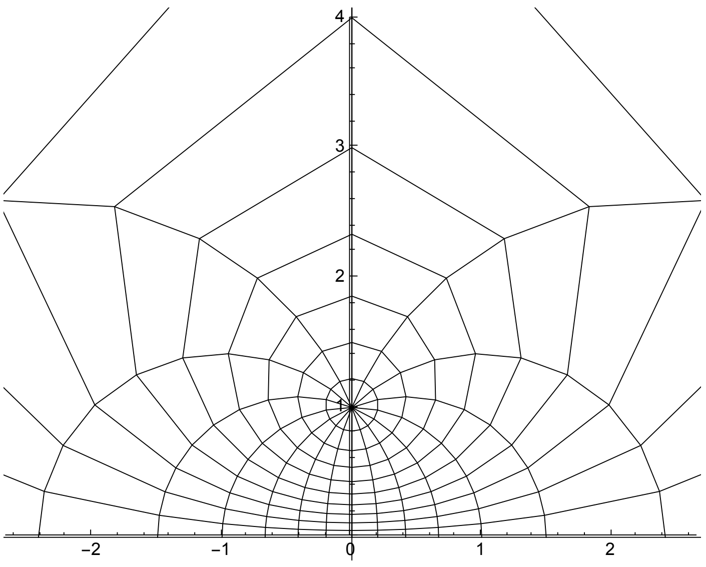
```
BeginPackage["MyComplexMap`"]

MyCartesianMap::usage="MyCartesianMap[f, {x0,x1,dx},{y0,y1,dy}]
   plots the image of the cartesian coordinate lines under
   the function  f."

MyPolarMap::usage="MyPolarMap[f, {r0,r1,dr}, {phi0,phi1,dphi}]
   plots the image of the polar coordinate lines under the
   function f."

Begin["`Private`"]

MakeLines[points_] :=
   Module[ {coords, lines},
      coords = Map[ {Re[#], Im[#]}&, points, {2} ];
      lines  = Map[ Line, Join[coords, Transpose[coords]] ];
      Graphics[lines]
      ]
MyCartesianMap[ func_, {x0_, x1_, dx_}, {y0_, y1_, dy_} ] :=
   Module[ {x, y, coords},
      coords = Table[ N[func[x + I y]], {x, x0, x1, dx},
                                        {y, y0, y1, dy}  ];
      Show[ MakeLines[coords], AspectRatio -> Automatic,
                               Axes        -> Automatic  ]
      ]

MyPolarMap[ func_, {r0_, r1_, dr_}, {phi0_, phi1_, dphi_}] :=
   Module[ {r, phi, coords},
   coords = Table[ N[func[ r Exp[I phi]]], {r, r0, r1, dr},
                                  {phi, phi0, phi1, dphi}];
   Show[ MakeLines[coords], AspectRatio -> Automatic,
                            Axes        -> Automatic  ]
   ]

End[(* "`Private`" *)]
EndPackage[]
```

```
MyComplexMap`

MyCartesianMap::shdw:
  Symbol MyCartesianMap appears in multiple contexts {MyComplexMap`, MyCartesianMap`}; definitions
     in context MyComplexMap` may shadow or be shadowed by other definitions. >>

MyCartesianMap[f, {x0,x1,dx},{y0,y1,dy}]
   plots the image of the cartesian coordinate lines under
   the function  f.

MyPolarMap[f, {r0,r1,dr}, {phi0,phi1,dphi}]
   plots the image of the polar coordinate lines under the
   function f.

MyComplexMap`Private`

MyComplexMap`Private`

MyCartesianMap::"shdw" :
   Symbol MyCartesianMap appears in multiple contexts {MyComplexMap`, MyCartesianMap`};
  definitions in context MyComplexMap` may shadow or be shadowed by other definitions. >>

MyComplexMap`

MyCartesianMap[f, {x0,x1,dx},{y0,y1,dy}]
   plots the image of the cartesian coordinate lines under
   the function  f.

MyPolarMap[f, {r0,r1,dr}, {phi0,phi1,dphi}]
   plots the image of the polar coordinate lines under the
   function f.

MyComplexMap`Private`

MyComplexMap`Private`
```

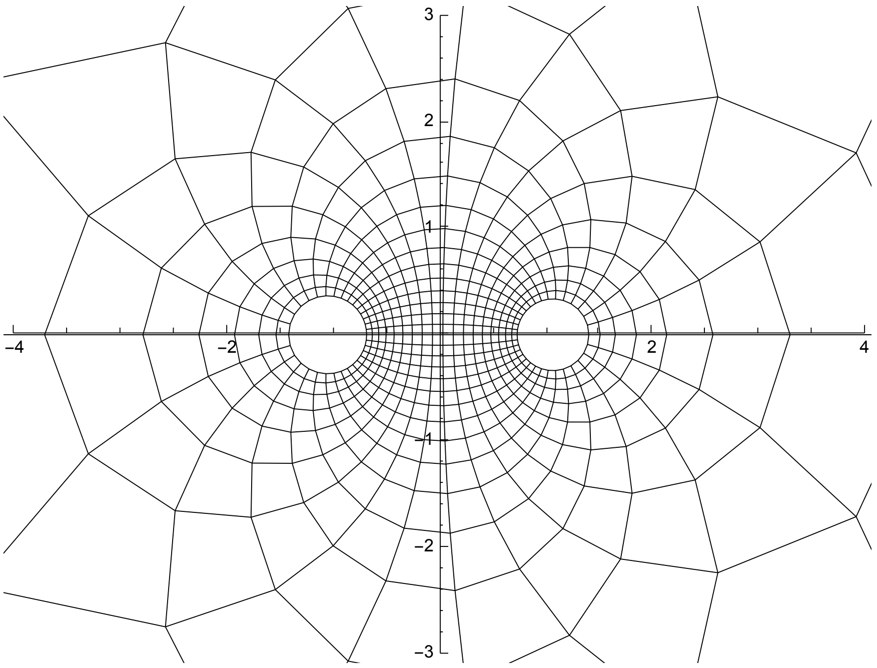
```
MyPolarMap[ # &, {0,1,.1}, {-Pi,Pi,Pi/8} ](* gives same figure as above.  *)
```



```
mp = MyPolarMap[-I (# + 1) / (# - 1) &, {0.0001, 1 + 0.0001, .1}, {-Pi, Pi, Pi/8} ];
Show[mp, PlotRange → {0, 4}]
```



```
ny = MyCartesianMap[ (Exp[#] - 1)/(Exp[#] + 1) &,
    {-1.1 Pi/2,1.15 Pi/2,Pi/16}, {-Pi,Pi,Pi/16} ];
Show[ny, PlotRange  -> {4{-1,1},3{-1,1}}, ImageSize ->  450]
```



```
Power::"infy" :  Infinite expression 1\0 encountered. >>
```

23.3 The Replacement of the Package "ComplexMap"

The new two-parameter form of `ParametricPlot` now provides the functionality of ``Graphics`ComplexMap` ``.

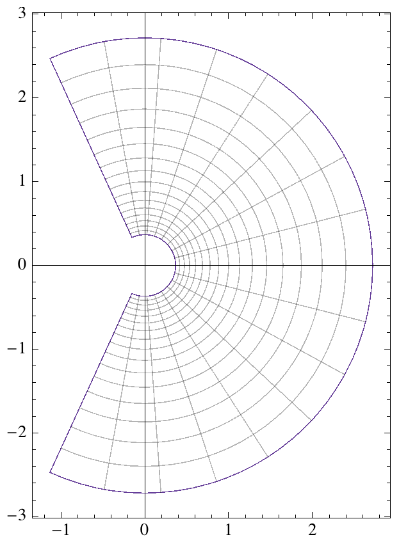
`CartesianMap` functionality is now produced using `ParametricPlot`:

```
?? Through
```

Through[*p*[*f*₁, *f*₂][*x*]] gives *p*[*f*₁[*x*], *f*₂[*x*]].
Through[*expr*, *h*] performs the transformation wherever *h* occurs in the head of *expr*. >>

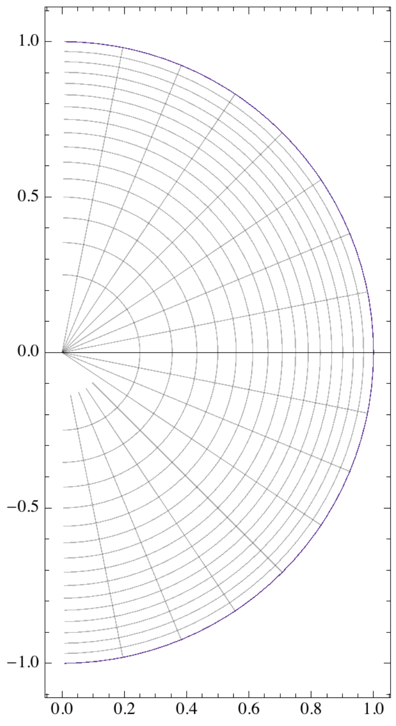
```
Attributes[Through] = {Protected}
```

```
ParametricPlot[Through[{Re, Im}[Exp[x + I * y]]], {x, -1, 1},
 {y, -2, 2}, PlotStyle → None, PlotTheme → Classic, ImageSize →  200]
```



`PolarMap` functionality is also produced using `ParametricPlot`:

```
ParametricPlot[Through[{Re, Im}[Sqrt[r Exp[I t]]]], {r, 0, 1},
 {t, 0, 2 Pi}, PlotStyle → None, PlotTheme → Classic, ImageSize →  200]
```



23.3 .1 Conformal maps

The package **Graphics`ConformalMaps`** of earlier versions of *Mathematica* and now the command **ParametricPlot[]** can be used to plot 2-dimensional fields obtained by conformal maps. How the complex
mapping function can be found is described in refs.[1,2,3]. For many configurations the mapping functions
can be found in tables, refs.[1,4].

1. G. Wendt, Statische Felder und Stationäre Ströme, pp. 56 - 60, in:
   Handbuch der Physik (Ed. S. Flügge), Bd.**16**, Springer, 1958.

2. J. van Bladel, Electromagnetic Fields, §§ 5.7, 5.8, McGraw-Hill, 1964.

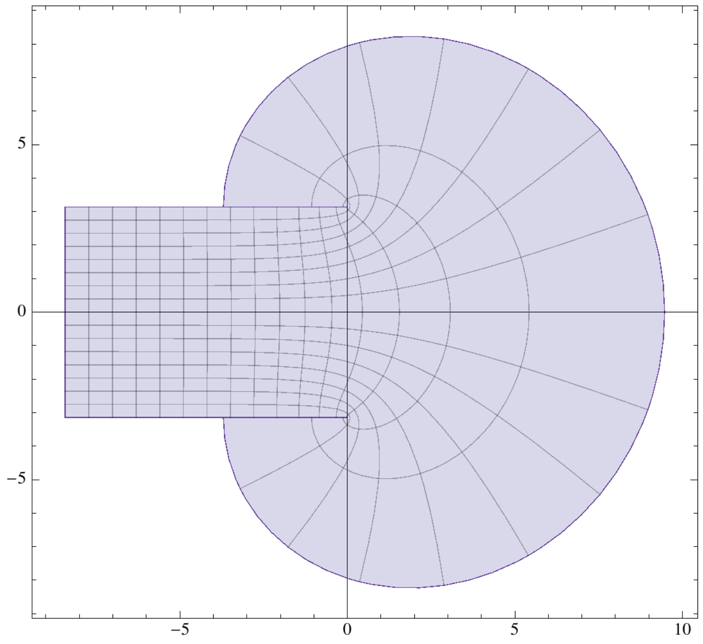
3. B. Schnizer, Analytische Methoden der Theoretischen Physik, Vorlesungskriptum, §13.10.
   http://itp.tugraz.at/~schnizer/AnalyticalMethods/

4. H. Kober, Dictionary of Conformal Representations. Dover, 1957.

23.3.2 The endfield of a semi-infinte plane condensor

```
Clear[fc]
fc[x_, y_] =  x + I y + 1 + Exp[x + I y] ;
```

```
ParametricPlot[Through[{Re, Im}[fc[x, y] ]],
 {x, - 3 Pi, .6 Pi}, {y, -Pi, Pi}, PlotTheme → Classic ]
```

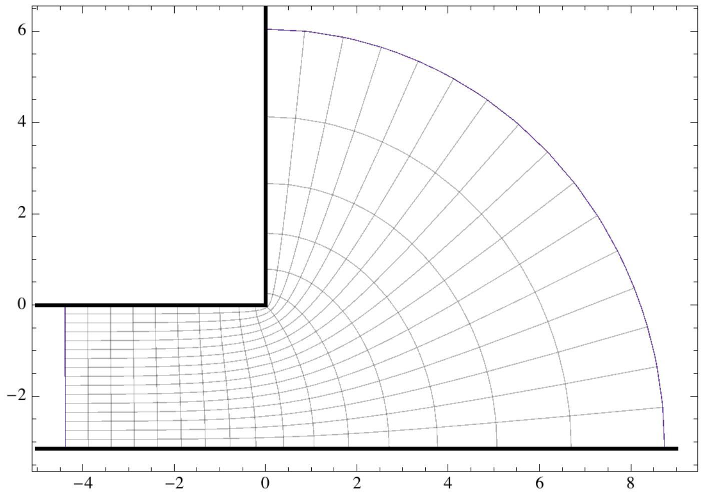


23.3.3 The endfield of a semi-infinte magnet pole (upper part only)

```
Clear[fc]
```

$$\texttt{fc[x\_, y\_] = -Conjugate}\left[2\,i\sqrt{\texttt{Exp[x + I y] - 1}} - \texttt{Log}\left[\frac{1 + i\sqrt{\texttt{Exp[x + I y] - 1}}}{1 - i\sqrt{\texttt{Exp[x + I y] - 1}}}\right]\right];$$

```
c1 = ListPlot[{{-5, 0}, {0, 0}, {0, 7}},
    Joined → True, PlotStyle → {Black, Thick} ];
c2 = ListPlot[N[{{-5, -π}, {9, -π}}], Joined → True, PlotStyle → {Black, Thick}];

ParametricPlot[Through[{Re, Im}[ fc[x, y] ]], {x, - 5, 3},
  {y, 0.0001`, Pi} , PlotStyle → None, PlotTheme → Classic, Axes → False ];

Show[%, c1, c2, Axes -> None]
```



The full endfield can be obtained by reflection on the horizontal symmetry plane (s. Ex.23.x) .

23.3.4 Failures of ParametricPlot[]

Unfortunately, **ParametricPlot[]** encounters troubles with most of the complex functions used for complex maps. **ParametricPlot[]** works satisfactorily in version 7 of *Mathematica;* but not with newer versions including *Mathematica10.*
You can call this by the command mathematica7 in our institute and computer rooms. An example

of the required input in a notebook is:

```
gd = 5; b = 8;
ParametricPlot[Through[
   {Re, Im}[Log[Cosh[Pi (x + I y + b / 2) / (2 gd)] / Cosh[Pi (x + I y - b / 2) / (2 gd)]]]],
  {x, -7.2, 7.2}, {y, 0.01, 4.98},  PlotStyle → None,
  PlotPoints → 150, Axes → True, FrameTicks → False,
  FrameLabel → {"x", None, None, "I  y"}, RotateLabel → False, ImageSize → 450,
  Epilog → {Thickness[0.005], Line[{{-8, Pi}, {8, Pi}}], Line[{{-8, 0}, {-2.5, 0}}],
    Line[{{8, 0}, {2.5, 0}}], GrayLevel[0.6], Line[{{-2.5, .0}, {2.5, 0}}]}]
```

23.4 Exercise

23. 1 The full field at end of a two - dimensional semi - infinite magnet.
The field between and outside the configurationn is given in § 23.3 .3 of notebook mathy23.nb.
By shifts and a reflection one can get a drawing for the full field.