

# ComplexFocus: Non-paraxial vector beams in Mathematica

© Rodrigo Gutiérrez-Cuevas & Emilio Pisanty (2020). Licensed under GPL and CC BY-SA.

This notebook generates the package file for the ComplexFocus package. For updates and additional information, go to <https://github.com/ComplexFocus/ComplexFocus>.

```
In[ ]:= (*
This is the ComplexFocus package, © Rodrigo Gutiérrez-Cuevas & Emilio Pisanty
(2020). For the notebook that generated this package file and additional documentaion,
see https://github.com/ComplexFocus/ComplexFocus.
*)
```

## Readme

### Specifications

The ComplexFocus package implements Complex Focus (CF) vector-beam fields as described in the papers

- Scalar and electromagnetic nonparaxial bases composed as superpositions of simple vortex fields with complex foci. R. Gutiérrez-Cuevas and M.A. Alonso. *Optics Express* **25**, pp. 14856-14870 (2017).
- Optical polarization skyrmionic fields in free space. R. Gutiérrez-Cuevas and E. Pisanty. Under review (2020).

These fields are built on the idea that (i) the multipolar spherical-wave functions,  $\Lambda_{l,m}(x, y, z) = j_l(r) Y_{l,m}(\theta, \phi)$ , are solutions of the Helmholtz equation,  $(\nabla^2 + 1)\Lambda = 0$ , and (ii) they retain this property if one of the variables is shifted by an imaginary-valued offset. This transformation thus keeps them as solutions of the wave equation, while at the same time turning them into focused beam-like solutions and eventually, in the large-offset limit, into Gaussian beam functions. The package thus implements:

- The multipolar functions  $\Lambda_{l,m}$ , as `MultipoleLambda`;
- The vector differential operators  $V_r^{(E)}(v, f) = \nabla \times (\nabla \times (v f))$  and  $V_r^{(M)}(v, f) = -i \nabla \times (v f)$ , of electric and magnetic type for a vector  $v$  and a scalar function  $f$ , as well as the helicity-type operator  $V_r^{(\pm)}(f) = V_r^{(E)}(e_{\pm}, f) \pm i V_r^{(M)}(e_{\pm}, f)$ , which lift the scalar solutions  $\Lambda_{l,m}$  into transverse vector solutions; and
- The complex-focus field functions that arise from this process.

### Implementation notes

To implement CF fields cleanly, we make heavy use of Mathematica's symbolic differentiator, working in Cartesian coordinates. If done directly, this poses a problem, since the Cartesian derivatives of the spherical harmonic  $Y_{l,m}(\theta, \phi)$  are complex combinations and present significantly singular behaviour at the origin – a property which is at odds with the fact that  $\Lambda_{l,m}(x, y, z)$  is an entire function of all three Cartesian coordinates.

To solve this problem, we re-factor the fundamental multipole functions as

$\Lambda_{l,m}(x, y, z) = j_l(r) Y_{l,m}(\theta, \phi) = a_{jl}(r) S_{l,m}(x, y, z)$ , where  $S_{l,m}(x, y, z) = r^l Y_{l,m}(\theta, \phi)$  are solid harmonics (and therefore homogeneous polynomials in  $x, y, z$ ) and  $a_{jl}(x, y, z) = \frac{1}{r^l} j_l(r) = \frac{1}{(2l+1)!!} - \frac{x^2+y^2+z^2}{2(2l+3)!!} + \dots$ , implemented as `AnalyticalBesselJ`, is the spherical Bessel function with its asymptotic behaviour at the origin removed (thus making it an even Taylor series in powers of  $x, y, z$ , without any square roots coming from  $r = \sqrt{x^2 + y^2 + z^2}$ ). The Cartesian derivatives of  $a_{jl}(x, y, z)$  are then coded explicitly using the recursion relation  $\frac{d}{dr} \left( \frac{1}{r^l} j_l(r) \right) = -\frac{1}{r^l} j_{l+1}(r)$  (DLMF 10.51.3) and the chain rule  $\frac{\partial r}{\partial x} = \frac{x}{r}$  to get  $\frac{\partial}{\partial x} a_{jl}(x, y, z) = -x a_{j,l+1}(x, y, z)$  (and equivalently for  $y$  and  $z$ ).

## Package admin

### Initialization

```
In[ ]:= BeginPackage["ComplexFocus`"];
```

### Package admin

#### Version managing

The variable `$ComplexFocusVersion` gives the version of the `ComplexFocus` package currently loaded as well as its timestamp

```
In[ ]:= $ComplexFocusVersion::usage =
"$ComplexFocusVersion prints the current version of the ComplexFocus
package in use and its timestamp.";
$ComplexFocusTimestamp::usage = "$ComplexFocusTimestamp prints the timestamp
of the current version of the ComplexFocus package.";
```

The timestamp is updated every time this notebook is saved, via an appropriate notebook option which is set by the code below.

```
SetOptions[
  EvaluationNotebook[],
  NotebookEventActions -> {{"MenuCommand", "Save"} -> (
    NotebookWrite[
      Cells[CellTags -> "version-timestamp"][[1]],
      Cell[
        BoxData[RowBox[{"$ComplexFocusTimestamp=" <> DateString[] <> "\";\\nEnd[];"}]]
        , "Input", InitializationCell -> True, CellTags -> "version-timestamp"
      ], None, AutoScroll -> False];
    NotebookSave[]
  ), PassEventsDown -> True}
];
```

To reset this behaviour to normal, evaluate the cell below

```
SetOptions[EvaluationNotebook[],
  NotebookEventActions -> {{"MenuCommand", "Save"} -> (NotebookSave[]), PassEventsDown -> True}]
```

## Version number and timestamp

```
In[ ]:= Begin["`Private`"];
$ComplexFocusVersion := "ComplexFocus v1.0, " <> $ComplexFocusTimestamp;

$ComplexFocusTimestamp = "Tue 1 Dec 2020 22:39:54";
End[];
```

## Directory

```
In[ ]:= $ComplexFocusDirectory::usage = "$ComplexFocusDirectory is the
directory where the current ComplexFocus package instance is located.";

In[ ]:= Begin["`Private`"];
With[{softLinkTestString = StringSplit[StringJoin[
    ReadList["! ls -la " <> StringReplace[$InputFileName, {" " -> "\\ "}], String]], " -> "]},
    If[Length[softLinkTestString] > 1, (*Testing in case $InputFileName
    is a soft link to the actual directory.*)
        $ComplexFocusDirectory = StringReplace[DirectoryName[softLinkTestString[[2]], {" " -> "\\ "}],
        $ComplexFocusDirectory = StringReplace[DirectoryName[$InputFileName], {" " -> "\\ "}],
    ]];
End[];
```

## Git commit hash and message

```
In[ ]:= $ComplexFocusCommit::usage =
"$ComplexFocusCommit returns the git commit log at the location of the
ComplexFocus package if there is one.";
$ComplexFocusCommit::OS = "$ComplexFocusCommit has only been tested on Linux.";

In[ ]:= Begin["`Private`"];
$ComplexFocusCommit := (If[$OperatingSystem != "Unix", Message[$ComplexFocusCommit::OS]];
    StringJoin[
        Riffle[ReadList["!cd " <> $ComplexFocusDirectory <> " && git log -1", String], {"\n"}]]);
End[];
```

# Package code

## SolidHarmonicS

This function implements the solid harmonic  $S_{l,m}(r) = r^l Y_{l,m}(\theta, \phi)$ , which is a homogeneous polynomial of degree  $l$ , and lends itself much better to symbolic differentiation than explicit spherical harmonics.

Code provided by J.M. at <http://mathematica.stackexchange.com/a/124336/1000> under the WTFPL.

```

In[ ]:= SolidHarmonicS::usage =
  "SolidHarmonicS[l,m,x,y,z] calculates the solid harmonic  $S_{lm}(x,y,z)=r^l Y_{lm}(x,y,z)$ .

  SolidHarmonicS[l,m,{x,y,z}] does the same.";
Begin["`Private`"];
SolidHarmonicS[λ_Integer, μ_Integer, x_, y_, z_] /; λ ≥ Abs[μ] :=
  Sqrt[ $\frac{2\lambda+1}{4\pi}$ ] Sqrt[ $\frac{\Gamma[\lambda - \text{Abs}[\mu] + 1]}{\Gamma[\lambda + \text{Abs}[\mu] + 1]}$ ]  $2^{-\lambda} (-1)^{(\mu - \text{Abs}[\mu])/2}$  ×
  If[Rationalize[μ] == 0, 1, (x + Sign[μ] i y)Abs[μ]] ×
  Sum[
    (-1)μ+k Binomial[λ, k] Binomial[2λ - 2k, λ] Pochhammer[λ - Abs[μ] - 2k + 1, Abs[μ]] ×
    If[TrueQ[Pochhammer[λ - Abs[μ] - 2k + 1, Abs[μ]] == 0], 1,
      If[Rationalize[k] == 0, 1, (x2 + y2 + z2)k] × If[Rationalize[λ - Abs[μ] - 2k] == 0, 1, zλ - Abs[μ] - 2k]
    ], {k, 0, Quotient[λ, 2]}]
SolidHarmonicS[λ_Integer, μ_Integer, {x_, y_, z_}] /; λ ≥ Abs[μ] := SolidHarmonicS[λ, μ, x, y, z]
End[];

```

## AnalyticalBesselJ

Define the analytical Bessel functions that go with the solid harmonics, as well as their differentiation rules.

```

In[ ]:= AnalyticalBesselJ::usage = "AnalyticalBesselJ[m,{x,y,z}] gives
  the analytical form of the spherical Bessel function,  $aj_m(r)=j_m(r)/r^m$ .";

Begin["`Private`"];
AnalyticalBesselJ[m_, {x_?NumericQ, y_?NumericQ, z_?NumericQ}] :=
   $\frac{\text{SphericalBesselJ}[\text{Abs}[m], \sqrt{x^2 + y^2 + z^2}]}{(x^2 + y^2 + z^2)^{\text{Abs}[m]/2}}$ 
AnalyticalBesselJ[m_, {x_?NumericQ, y_?NumericQ, z_?NumericQ}] /;
  (x2 + y2 + z2 == 0 || x2 + y2 + z2 == 0.)] :=  $\frac{1}{(2m+1)!!}$ 

Derivative[0, {1, 0, 0}][AnalyticalBesselJ][m_, {x_, y_, z_}] :=
  (-1) x AnalyticalBesselJ[m+1, {x, y, z}]
Derivative[0, {0, 1, 0}][AnalyticalBesselJ][m_, {x_, y_, z_}] :=
  (-1) y AnalyticalBesselJ[m+1, {x, y, z}]
Derivative[0, {0, 0, 1}][AnalyticalBesselJ][m_, {x_, y_, z_}] :=
  (-1) z AnalyticalBesselJ[m+1, {x, y, z}]

End[];

```

## MultipoleΛ

```
In[ ]:= MultipoleΛ::usage = "MultipoleΛ[l,m,{x,y,z}] calculates the
      multipole function  $\Lambda_{l,m}(x,y,z) = 4\pi i^l j_l(r) Y_{lm}(\theta,\phi) = 4\pi i^l a_{jl}(r) S_{lm}(x,y,z)$  .";

Begin["`Private`"];

MultipoleΛ[l_, m_, {x_, y_, z_}] :=
  4 π i^l AnalyticalBesselJ[l, {x, y, z}] × SolidHarmonicS[l, m, x, y, z]

End[];
```

## Polarization handling

### Unit vectors

```
In[ ]:= UnitE::usage = "UnitE[s] gives the unit vectors  $e_s$  for
       $s=-1,0,1$ , equal to  $(1,-i,0)/\sqrt{2}$ ,  $(0,0,1)$  and  $(1,i,0)/\sqrt{2}$ , respectively.";
UnitU::usage = "UnitU[θ,φ] gives the radial unit vector
       $(\sin(\theta)\cos(\phi), \sin(\theta)\sin(\phi), \cos(\theta))$  at spherical polar coordinates  $(\theta,\phi)$  .";
Unitθ::usage = "Unitθ[θ,φ] gives the polar unit vector
       $(\cos(\theta)\cos(\phi), \cos(\theta)\sin(\phi), -\sin(\theta))$  at spherical polar coordinates  $(\theta,\phi)$  .";
Unitφ::usage = "Unitφ[θ,φ] gives the azimuthal unit vector  $(-\sin(\phi), \cos(\phi), 0)$ 
      at spherical polar coordinates  $(\theta,\phi)$  .";

Begin["`Private`"];

UnitE[s : (1 | -1)] :=  $\frac{1}{\sqrt{2}}$  {1, s i, 0}
UnitE[0] := {0, 0, 1}

UnitU[θ_, φ_] := {Cos[φ] Sin[θ], Sin[φ] Sin[θ], Cos[θ]}

Unitθ[θ_, φ_] := {Cos[φ] Cos[θ], Sin[φ] Cos[θ], -Sin[θ]}
Unitφ[θ_, φ_] := {-Sin[φ], Cos[φ], 0}

End[];
```

### PolarizationV variants (real space)

```
In[ ]:= PolarizationVElectric::usage =
  "PolarizationVElectric[v,f[x,y,z],{x,y,z}] gives the electric-type polarization
  operator  $V_v^{(E)}f = -\nabla \times (\nabla \times (v f))$  for a vector  $v$  and a scalar function  $f$ .
PolarizationVElectric[v,f] gives the functional form of  $V_v^{(E)}f$ . The vector
   $v$  can be a fixed object or an explicit Function object.";
PolarizationVMagnetic::usage = "PolarizationVMagnetic[v,f[x,y,z],{x,y,z}]
  gives the magnetic-type polarization operator
 $V_v^{(M)}f = -i\nabla \times (v f)$  for a vector  $v$  and a scalar function  $f$ .
PolarizationVMagnetic[v,f] gives the functional form of  $V_v^{(M)}f$ . The vector
```

```

v can be a fixed object or an explicit Function object.";
PolarizationVHelicity::usage = "PolarizationVHelicity[σ,f[x,y,z],{x,y,z}]
gives the helicity-type polarization operator
 $V^{(\pm)}f = V_{e_z}^{(E)}f \pm iV_{e_z}^{(M)}f$  for a helicity  $\sigma = \pm 1$  and a scalar function f.
PolarizationVHelicity[σ,f] gives the functional form of  $V^{(\pm)}f$ .";

Begin["`Private`"];

PolarizationVElectric[vector_, scalarFunction_, variables_] :=
- Curl[Curl[Times[vector, scalarFunction], variables], variables]
PolarizationVMagnetic[vector_, scalarFunction_, variables_] :=
- i Curl[Times[vector, scalarFunction], variables]
PolarizationVHelicity[s : (1 | -1), scalarFunction_, variables_] :=
PolarizationVElectric[UnitE[s], scalarFunction, variables] -
i s PolarizationVMagnetic[UnitE[s], scalarFunction, variables]

PolarizationVElectric[vector_, scalarFunction_] := Block[{x, y, z},
Function[{x, y, z}, Evaluate[
PolarizationVElectric[vector, scalarFunction[x, y, z], {x, y, z}]
]]
]
PolarizationVElectric[vector_Function, scalarFunction_] := Block[{x, y, z},
Function[{x, y, z}, Evaluate[
PolarizationVElectric[vector[x, y, z], scalarFunction[x, y, z], {x, y, z}]
]]
]

PolarizationVMagnetic[vector_, scalarFunction_] := Block[{x, y, z},
Function[{x, y, z}, Evaluate[
PolarizationVMagnetic[vector, scalarFunction[x, y, z], {x, y, z}]
]]
]
PolarizationVMagnetic[vector_Function, scalarFunction_] := Block[{x, y, z},
Function[{x, y, z}, Evaluate[
PolarizationVMagnetic[vector[x, y, z], scalarFunction[x, y, z], {x, y, z}]
]]
]

PolarizationVHelicity[s : (1 | -1), scalarFunction_] := Block[{x, y, z},
Function[{x, y, z}, Evaluate[
PolarizationVElectric[UnitE[s], scalarFunction[x, y, z], {x, y, z}] +
i s PolarizationVMagnetic[UnitE[s], scalarFunction[x, y, z], {x, y, z}]
]]
]

End[];

```

## Polarization $\mathcal{V}$ variants (PWS space)

The symbol  $\mathcal{V}$  can be typeset using the combination `scV`.

In[ ]:=

```

PolarizationVElectric::usage =
  "PolarizationVElectric[v,u] gives the electric-type polarization vector  $\mathcal{V}_v^{(E)} =$ 
     $u \times (u \times v)$  in Fourier space, in terms of the unit direction vector  $u = \{u_x, u_y, u_z\}$ .
  PolarizationVElectric[v, $\theta$ , $\phi$ ] gives the electric-type polarization vector  $\mathcal{V}_v^{(E)} = u \times (u \times v)$  in
    Fourier space, in terms of the polar coordinates  $(\theta, \phi)$  of the unit direction vector  $u$ .";

PolarizationVMagnetic::usage =
  "PolarizationVMagnetic[v,u] gives the electric-type polarization vector  $\mathcal{V}_v^{(E)} =$ 
     $u \times v$  in Fourier space, in terms of the unit direction vector  $u = \{u_x, u_y, u_z\}$ .
  PolarizationVMagnetic[v, $\theta$ , $\phi$ ] gives the electric-type polarization vector  $\mathcal{V}_v^{(E)} = u \times v$  in
    Fourier space, in terms of the polar coordinates  $(\theta, \phi)$  of the unit direction vector  $u$ .";

PolarizationVHelicity::usage =
  "PolarizationVHelicity[v,u] gives the electric-type polarization vector  $\mathcal{V}_v^{(E)} =$ 
     $u \times v$  in Fourier space, in terms of the unit direction vector  $u = \{u_x, u_y, u_z\}$ .
  PolarizationVHelicity[v, $\theta$ , $\phi$ ] gives the electric-type polarization vector  $\mathcal{V}_v^{(E)} = u \times v$  in
    Fourier space, in terms of the polar coordinates  $(\theta, \phi)$  of the unit direction vector  $u$ .";

Begin["`Private`"];

PolarizationVElectric[v_List, u_List] := Cross[u, Cross[u, v]]
PolarizationVElectric[v_List,  $\theta$ _,  $\phi$ _] := Cross[UnitU[ $\theta$ ,  $\phi$ ], Cross[UnitU[ $\theta$ ,  $\phi$ ], v]]

PolarizationVMagnetic[v_List, u_List] := Cross[u, v]
PolarizationVMagnetic[v_List,  $\theta$ _,  $\phi$ _] := Cross[UnitU[ $\theta$ ,  $\phi$ ], v]

PolarizationVHelicity[s : {1 | -1}, u_List] :=
  PolarizationVElectric[UnitE[s], u] -  $\mathfrak{i}$  s PolarizationVMagnetic[UnitE[s], u]
PolarizationVHelicity[s : {1 | -1},  $\theta$ _,  $\phi$ _] :=
  PolarizationVElectric[UnitE[s], UnitU[ $\theta$ ,  $\phi$ ]] -  $\mathfrak{i}$  s PolarizationVMagnetic[UnitE[s], UnitU[ $\theta$ ,  $\phi$ ]]

End[];

```

## Normalization

### CFnormTEM

Using partial memoization as per <https://mathematica.stackexchange.com/q/21782>

```

In[ ]:= CFnormTEM::usage = "CFnormTEM[l,m,q] returns the normalization
integral  $\int |Y_{lm}(\theta, \phi)|^2 e^{2q \cos(\theta)} \sin(\theta)^2 d\Omega$  for the TEM complex-focus fields.";

 $\alpha$ TEM::usage = " $\alpha$ TEM[l,m,q] returns the normalization
constant CFnormTEM[l,m,q]-1/2 for the TEM complex-focus fields.";

Begin["`Private`"];
 $\alpha$ TEM[l_, m_, q_] := CFnormTEM[l, m, q]-1/2
 $\alpha$ TEM[l_, m_, 0] :=  $\alpha$ TEM[l, m, 0] = CFnormTEM[l, m, 0]-1/2

CFnormTEM[l_, m_, q_] := Block[{qInt},
  CFnormTEM[l, m, qInt_] =
    Simplify[Integrate[SphericalHarmonicY[l, m,  $\theta$ ,  $\phi$ ] SphericalHarmonicY[l, -m,  $\theta$ ,  $\phi$ ]
      Exp[2 qInt Cos[ $\theta$ ]] (Sin[ $\theta$ ])2 Sin[ $\theta$ ], { $\theta$ , 0,  $\pi$ }, { $\phi$ , 0, 2  $\pi$ }]];
  CFnormTEM[l, m, q]
]
CFnormTEM[l_, m_, 0] := CFnormTEM[l, m, 0] = Limit[CFnormTEM[l, m, q], q  $\rightarrow$  0]
End[];

```

## Special cases of CFnormTEM

The partially-memoized evaluation in the definition of CFnormTEM ensures that the expensive symbolic integration is only ever performed once for each  $l, m$  pair. However, this still means that the code will take a few seconds to run the first time that each pair is evaluated.

This can be mitigated by pre-computing the first few cases and including them as explicit definitions in the package .m file. To do that, the code below computes the first few cases and then writes them into the initialization cell below.

```

NotebookWrite[
  Cells[CellTags  $\rightarrow$  "CFNormTEM-SpecialCases"][[1],
  Cell[BoxData[RowBox[Most[Flatten[Join[
    {"(*This cell is auto-generated. Any changes will be over-ridden
      if the cell above is evaluated.*)", "\[IndentingNewLine]"},
    {"Begin[\"`Private`\""];", "\[IndentingNewLine]"},
    Table[Table[{
      "CFnormTEM[" <> ToString[l] <> ", " <> ToString[m] <> ", q_]",
      ":",
      ToString[CFnormTEM[l, m, q], InputForm],
      "\[IndentingNewLine]"
    }, {m, -1, 1}], {l, 0, 3}],
    {"End[];", "\[IndentingNewLine]"}
  ]]]], "Input", InitializationCell  $\rightarrow$  True, CellTags  $\rightarrow$  "CFNormTEM-SpecialCases"
], None, AutoScroll  $\rightarrow$  False]

```



In[ ]:=

```

(*This cell is auto-generated. Any changes will be over-
ridden if the cell above is evaluated.*/)
Begin["`Private`"];
CFnormTEM[0, 0, q_] := (2 * q * Cosh[2 * q] - Sinh[2 * q]) / (4 * q^3)
CFnormTEM[1, -1, q_] := (18 * q * Cosh[2 * q] - 3 * (3 + 4 * q^2) * Sinh[2 * q]) / (8 * q^5)
CFnormTEM[1, 0, q_] := (6 * q * (3 + q^2) * Cosh[2 * q] - 3 * (3 + 5 * q^2) * Sinh[2 * q]) / (4 * q^5)
CFnormTEM[1, 1, q_] := (18 * q * Cosh[2 * q] - 3 * (3 + 4 * q^2) * Sinh[2 * q]) / (8 * q^5)
CFnormTEM[2, -2, q_] :=
  (45 * (2 * q * (15 + 4 * q^2) * Cosh[2 * q] - 3 * (5 + 8 * q^2) * Sinh[2 * q])) / (64 * q^7)
CFnormTEM[2, -1, q_] :=
  (15 * (18 * q * (5 + 2 * q^2) * Cosh[2 * q] - (45 + 78 * q^2 + 8 * q^4) * Sinh[2 * q])) / (16 * q^7)
CFnormTEM[2, 0, q_] :=
  (5 * (2 * q * (405 + 180 * q^2 + 8 * q^4) * Cosh[2 * q] - (405 + 720 * q^2 + 104 * q^4) * Sinh[2 * q])) /
  (32 * q^7)
CFnormTEM[2, 1, q_] :=
  (15 * (18 * q * (5 + 2 * q^2) * Cosh[2 * q] - (45 + 78 * q^2 + 8 * q^4) * Sinh[2 * q])) / (16 * q^7)
CFnormTEM[2, 2, q_] :=
  (45 * (2 * q * (15 + 4 * q^2) * Cosh[2 * q] - 3 * (5 + 8 * q^2) * Sinh[2 * q])) / (64 * q^7)
CFnormTEM[3, -3, q_] :=
  (105 * (10 * q * (21 + 8 * q^2) * Cosh[2 * q] - (105 + 180 * q^2 + 16 * q^4) * Sinh[2 * q])) / (64 * q^9)
CFnormTEM[3, -2, q_] :=
  (315 * (2 * q * (210 + 95 * q^2 + 4 * q^4) * Cosh[2 * q] - (210 + 375 * q^2 + 56 * q^4) * Sinh[2 * q])) /
  (64 * q^9)
CFnormTEM[3, -1, q_] := (21 * (6 * q * (2625 + 1300 * q^2 + 96 * q^4) * Cosh[2 * q] -
  (7875 + 64 * q^2 * (225 + 42 * q^2 + q^4)) * Sinh[2 * q])) / (64 * q^9)
CFnormTEM[3, 0, q_] := (7 * (2 * q * (15750 + 8025 * q^2 + 684 * q^4 + 8 * q^6) * Cosh[2 * q] -
  (15750 + 29025 * q^2 + 5784 * q^4 + 200 * q^6) * Sinh[2 * q])) / (32 * q^9)
CFnormTEM[3, 1, q_] := (21 * (6 * q * (2625 + 1300 * q^2 + 96 * q^4) * Cosh[2 * q] -
  (7875 + 64 * q^2 * (225 + 42 * q^2 + q^4)) * Sinh[2 * q])) / (64 * q^9)
CFnormTEM[3, 2, q_] := (315 * (2 * q * (210 + 95 * q^2 + 4 * q^4) * Cosh[2 * q] -
  (210 + 375 * q^2 + 56 * q^4) * Sinh[2 * q])) / (64 * q^9)
CFnormTEM[3, 3, q_] := (105 * (10 * q * (21 + 8 * q^2) * Cosh[2 * q] -
  (105 + 180 * q^2 + 16 * q^4) * Sinh[2 * q])) / (64 * q^9)
End[];

```

To kick-start the process, run `SetOptions[EvaluationCell[], CellTags → "sample-cell-tag"]` on the chosen cell with an appropriate cell tag.

## CFnormQuasiCircular

Using partial memoization as with CFnormTEM above.

```

In[ ]:= CFnormQuasiCircular::usage = "CFnormQuasiCircular[l,m,q] returns the normalization integral
 $\int |Y_{lm}(\theta, \phi)|^2 e^{2q \cos(\theta)} (1 + \cos(\theta))^2 d\Omega$  for the quasi-circular complex-focus fields.";

αQC::usage = "αQC[l,m,q] returns the normalization constant
CFnormQuasiCircular[l,m,q]^{-1/2} for the quasi-circular complex-focus fields.";

Begin["`Private`"];
αQC[l_, m_, q_] := CFnormQuasiCircular[l, m, q]^{-1/2}
αQC[l_, m_, 0] := αQC[l, m, 0] = CFnormQuasiCircular[l, m, 0]^{-1/2}

CFnormQuasiCircular[l_, m_, q_] := Block[{qInt},
  CFnormQuasiCircular[l, m, qInt_] =
    Simplify[Integrate[SphericalHarmonicY[l, m, θ, φ] SphericalHarmonicY[l, -m, θ, φ]
      Exp[2 qInt Cos[θ]] (1 + Cos[θ])^2 Sin[θ], {θ, 0, π}, {φ, 0, 2 π}]];
  CFnormQuasiCircular[l, m, q]
]
CFnormQuasiCircular[l_, m_, 0] :=
  CFnormQuasiCircular[l, m, 0] = Limit[CFnormQuasiCircular[l, m, q], q → 0]
End[];

```

## Special cases of CFnormQuasiCircular

Identical to the CFnormTEM special-case handling above, but for CFnormQuasiCircular.

```

NotebookWrite[
  Cells[CellTags → "CFnormQuasiCircular-SpecialCases"] [[1]],
  Cell[BoxData[RowBox[Most[Flatten[Join[
    {"(*This cell is auto-generated. Any changes will be over-ridden
      if the cell above is evaluated.*)", "\[IndentingNewLine]"},
    {"Begin[\"`Private`\""];", "\[IndentingNewLine]"},
    Table[Table[{
      "CFnormQuasiCircular[" <> ToString[l] <> ", " <> ToString[m] <> ", q_]",
      ":",
      ToString[CFnormQuasiCircular[l, m, q], InputForm],
      "\[IndentingNewLine]"
    }, {m, -1, 1}], {l, 0, 3}],
    {"End[];", "\[IndentingNewLine]"}
  ]]]], "Input", InitializationCell → True, CellTags → "CFnormQuasiCircular-SpecialCases"
], None, AutoScroll → False]

```

In[ ]:=

```

(*This cell is auto-generated. Any changes will be over-
ridden if the cell above is evaluated.*/)
Begin["`Private`"];
CFnormQuasiCircular[0, 0, q_] := (-1 + E^(4 * q) * (1 - 4 * q + 8 * q^2)) / (8 * E^(2 * q) * q^3)
CFnormQuasiCircular[1, -1, q_] :=
(-3 * (3 * (1 + q) + E^(4 * q) * (-3 + 9 * q - 12 * q^2 + 8 * q^3))) / (16 * E^(2 * q) * q^5)
CFnormQuasiCircular[1, 0, q_] :=
(3 * (-3 - 3 * q - q^2 + E^(4 * q) * (3 - 9 * q + 13 * q^2 - 12 * q^3 + 8 * q^4))) / (8 * E^(2 * q) * q^5)
CFnormQuasiCircular[1, 1, q_] :=
(-3 * (3 * (1 + q) + E^(4 * q) * (-3 + 9 * q - 12 * q^2 + 8 * q^3))) / (16 * E^(2 * q) * q^5)
CFnormQuasiCircular[2, -2, q_] :=
(15 * (-3 * (15 + 20 * q + 8 * q^2) + E^(4 * q) * (45 - 120 * q + 144 * q^2 - 96 * q^3 + 32 * q^4))) /
(128 * E^(2 * q) * q^7)
CFnormQuasiCircular[2, -1, q_] := (-15 * (45 + 60 * q + 30 * q^2 + 6 * q^3 +
E^(4 * q) * (-45 + 120 * q - 150 * q^2 + 114 * q^3 - 56 * q^4 + 16 * q^5))) / (32 * E^(2 * q) * q^7)
CFnormQuasiCircular[2, 0, q_] := (5 * (-405 - 4 * q * (135 + 2 * q * (36 + q * (9 + q)))) + E^(4 * q) *
(405 + 8 * q * (-135 + q * (171 + q * (-135 + q * (73 + 4 * q * (-7 + 2 * q))))))) / (64 * E^(2 *
q) * q^7)
CFnormQuasiCircular[2, 1, q_] := (-15 * (45 + 60 * q + 30 * q^2 + 6 * q^3 +
E^(4 * q) * (-45 + 120 * q - 150 * q^2 + 114 * q^3 - 56 * q^4 + 16 * q^5))) / (32 * E^(2 * q) * q^7)
CFnormQuasiCircular[2, 2, q_] := (15 * (-3 * (15 + 20 * q + 8 * q^2) +
E^(4 * q) * (45 - 120 * q + 144 * q^2 - 96 * q^3 + 32 * q^4))) / (128 * E^(2 * q) * q^7)
CFnormQuasiCircular[3, -3, q_] := (-525 * (42 + q * (63 + 4 * q * (9 + 2 * q))) - 105 * E^(4 * q) *
(-210 + q * (525 + 8 * q * (-75 + 2 * q * (25 + 2 * (-5 + q) * q)))))) / (256 * E^(2 * q) * q^9)
CFnormQuasiCircular[3, -2, q_] := (105 * (-3 * (210 + q * (315 + q * (195 + 60 * q + 8 * q^2))) +
E^(4 * q) * (630 + q * (-1575 + q * (1845 + 8 * q * (-165 + 2 * q * (39 + 2 * (-6 + q) * q))))))) /
(128 * E^(2 * q) * q^9)
CFnormQuasiCircular[3, -1, q_] :=
(-63 * (5250 + q * (7875 + 4 * q * (1275 + 2 * q * (225 + 4 * q * (11 + q)))))) -
21 * E^(4 * q) * (-15750 + q * (39375 + 16 * q * (-2925 +
q * (2175 + 4 * q * (-279 + 2 * q * (51 + q * (-13 + 2 * q))))))) / (256 * E^(2 * q) * q^9)
CFnormQuasiCircular[3, 0, q_] := (-55125 * (2 + 3 * q) - 7 * q^2 *
(15525 + 4 * q * (1425 + 2 * q * (153 + q * (18 + q)))) +
7 * E^(4 * q) * (15750 + q * (-39375 + q * (47025 + 8 * q * (-4425 +
q * (2328 + q * (-894 + q * (253 - 52 * q + 8 * q^2))))))) / (64 * E^(2 * q) * q^9)
CFnormQuasiCircular[3, 1, q_] := (-63 * (5250 + q * (7875 + 4 * q *
(1275 + 2 * q * (225 + 4 * q * (11 + q)))))) -
21 * E^(4 * q) * (-15750 + q * (39375 + 16 * q * (-2925 + q * (2175 + 4 * q *
(-279 + 2 * q * (51 + q * (-13 + 2 * q))))))) / (256 * E^(2 * q) * q^9)
CFnormQuasiCircular[3, 2, q_] := (105 * (-3 * (210 + q * (315 + q * (195 + 60 * q + 8 * q^2))) +
E^(4 * q) * (630 + q * (-1575 + q * (1845 + 8 * q * (-165 + 2 * q * (39 + 2 * (-6 + q) * q))))))) /
(128 * E^(2 * q) * q^9)
CFnormQuasiCircular[3, 3, q_] := (-525 * (42 + q * (63 + 4 * q * (9 + 2 * q))) - 105 * E^(4 * q) *
(-210 + q * (525 + 8 * q * (-75 + 2 * q * (25 + 2 * (-5 + q) * q)))))) / (256 * E^(2 * q) * q^9)
End[];

```

## Fields

### ComplexFocusHelicityE, ComplexFocusElectricE, ComplexFocusMagneticE

```
In[8]:= ComplexFocusHelicityE::usage =
  "ComplexFocusHelicityE[l,m,σ,{x,y,z},ξ] gives the complex-focus field of helicity type,
  with complex-focus parameter ξ, angular momentum numbers l,m and helicity σ=±1.";
ComplexFocusElectricE::usage = "ComplexFocusElectricE[l,m,v,{x,y,z},ξ] gives
  the complex-focus field of electric type, with complex-focus parameter
  ξ, angular momentum numbers l,m and seed polarization vector v.";
ComplexFocusMagneticE::usage = "ComplexFocusMagneticE[l,m,v,{x,y,z},ξ] gives
  the complex-focus field of magnetic type, with complex-focus parameter
  ξ, angular momentum numbers l,m and seed polarization vector v.";

Begin["`Private`"];
ComplexFocusHelicityE[l_, m_, s : (1 | -1), {x_, y_, z_}, ξ_] := Block[{xInt, yInt, zInt, ξInt},
  ComplexFocusHelicityE[l, m, s, {xInt_, yInt_, zInt_}, ξInt_] = Simplify[
    PolarizationVHelicity[s, MultipoleΛ[l, m, {xInt, yInt, zInt - i ξInt}], {xInt, yInt, zInt}]];
ComplexFocusHelicityE[l, m, s, {x, y, z}, ξ]
]
ComplexFocusElectricE[l_, m_, vector_, {x_, y_, z_}, ξ_] := Block[{xInt, yInt, zInt, ξInt},
  ComplexFocusElectricE[l, m, vector, {xInt_, yInt_, zInt_}, ξInt_] =
    Simplify[PolarizationVElectric[vector,
      MultipoleΛ[l, m, {xInt, yInt, zInt - i ξInt}], {xInt, yInt, zInt}]];
ComplexFocusElectricE[l, m, vector, {x, y, z}, ξ]
]
ComplexFocusMagneticE[l_, m_, vector_, {x_, y_, z_}, ξ_] := Block[{xInt, yInt, zInt, ξInt},
  ComplexFocusMagneticE[l, m, vector, {xInt_, yInt_, zInt_}, ξInt_] =
    Simplify[PolarizationVMagnetic[vector,
      MultipoleΛ[l, m, {xInt, yInt, zInt - i ξInt}], {xInt, yInt, zInt}]];
ComplexFocusMagneticE[l, m, vector, {x, y, z}, ξ]
]
End[];
```

## Polarization analysis functions

### JonesPhase

```
In[9]:= JonesPhase::usage = "JonesPhase[v] gives the Jones phase of the vector v,  $\frac{1}{2}\arg(v \cdot v)$ .";

Begin["`Private`"];

JonesPhase[vec_] := Arg[Dot[vec, vec]] / 2;

End[];
```

## JonesMajorAxisA

```
In[ ]:= JonesMajorAxisA::usage = "JonesMajorAxisA[v] gives the major-axis
      vector of the polarization ellipse marked by the complex vector v.
      JonesMajorAxisA[v,normFunction] normalizes the ellipse by normFunction[v].";

Begin["`Private`"];

JonesMajorAxisA[vec_, normFunction_ : (1 &)] := Block[{vecF},
  vecF = vec Exp[-I JonesPhase[vec]];
  Re[vecF] / normFunction[vecF]
]

End[];
```

## JonesMinorAxisB

```
In[ ]:= JonesMinorAxisB::usage = "JonesMinorAxisB[v] gives the minor-axis
      vector of the polarization ellipse marked by the complex vector v.
      JonesMinorAxisB[v,normFunction] normalizes the ellipse by normFunction[v].";

Begin["`Private`"];

JonesMinorAxisB[vec_, normFunction_ : (1 &)] := Block[{vecF},
  vecF = vec Exp[-I JonesPhase[vec]];
  Im[vecF] / normFunction[vecF]
]

End[];
```

## SpinE

```
In[ ]:= SpinE::usage = "SpinE[E] gives the electric spin angular
      momentum vector  $\frac{1}{|E|^2} \text{Im}(E^* \times E)$  for a complex electric-field amplitude E.";

Begin["`Private`"];

SpinE[vec_] := Im[Cross[Conjugate[vec], vec]] / Norm[vec]^2

End[];
```

## Poincarana

Poincarana description of the polarization, as described in

Geometric phases in 2D and 3D polarized fields: geometrical, dynamical, and topological aspects. KY Bliokh, MA Alonso and MR Dennis. *Rep. Prog. Phys.* **82**, 122401 (2019), HAL:02342161.

In[ ]:=

```

Poincarana::usage =
  "Poincarana[E] gives the Poincarana-representation vectors  $\{\vec{u}_1, \vec{u}_2\}$  for the electric-field
  complex amplitude  $E = \vec{E} = \{E_x, E_y, E_z\} = e^{i\phi} (\vec{A} + i\vec{B})$  (in the Jones decomposition),
  given by  $\vec{u}_i = \pm f^2 \vec{a} + \vec{S}$ , where  $\vec{S} = \text{Im}(\vec{E}^* \times \vec{E}) / |\vec{E}|^2$  is the spin angular
  momentum,  $f^2 = |\vec{A}| - |\vec{B}|$  is the focal distance of the intensity-normalized
  ellipse, and  $\vec{a} = \vec{A} / |\vec{A}|$  is the normalized major-axis direction.";

Begin["`Private`"];

Poincarana[vectorE_] := Module[{vectorF, comp,  $\alpha$ ,  $\beta$ , avec, jvec},
  vectorF = -Exp[-I JonesPhase[vectorE]] vectorE;
   $\alpha$  = Chop[vectorF.vectorF / Norm[vectorF]^2];
   $\beta$  = Sqrt[1 -  $\alpha^2$ ];
  avec = Re[vectorF] / Norm[Re[vectorF]];
  jvec = If[Norm[Im[vectorF]] == 0, 0, Cross[avec, Im[vectorF] / Norm[Im[vectorF]]]];

  { $\alpha$  avec +  $\beta$  jvec, - $\alpha$  avec +  $\beta$  jvec}
]

End[];

```

## Visualization functions

### UniformRadialGrid

In[ ]:=

```

UniformRadialGrid::usage = "UniformRadialGrid[n] produces a uniform radial grid with n rings.
UniformRadialGrid[n,m] produces a
  uniform radial grid with n rings and m points on the first ring.
UniformRadialGrid[n,m, $\Delta r$ ] produces a uniform radial grid with n rings, m
  points on the first ring, and spacing of  $\Delta r$  between the rings.";

Begin["`Private`"];

UniformRadialGrid[nRings_?IntegerQ, nCircle_?IntegerQ,  $\Delta r$ _] := Times[ $\Delta r$ ,
  Join[{0, 0},
    Flatten[
      Table[
        Table[
          {n Cos[ $\phi$ ], n Sin[ $\phi$ ]}
          , { $\phi$ , Most[Subdivide[0, 2  $\pi$ , (nCircle - 1) n + 1]]}]
        , {n, 1, nRings}]
      , 1]
    ]
  ]

UniformRadialGrid[nRings_?IntegerQ, nCircle_?IntegerQ] := UniformRadialGrid[nRings, nCircle, 1]
UniformRadialGrid[nRings_?IntegerQ] := UniformRadialGrid[nRings, 7, 1]

End[];

```

This requires the function `Subdivide`, which dates from v10.1, so here is a fill-in substitute if running from v10.0 or earlier:

*In[ ]:=*

```
If[
  $VersionNumber < 10.1,

  Subdivide::usage =
    "\!\(\*RowBox[{\"Subdivide\", \"[\", StyleBox[\"n\", \"TI\"], \"]\"}]\) generates
    the list \!\(\*RowBox[{\"{\", RowBox[{\"0\", \"\", RowBox[{\"1\", \"/\",
    StyleBox[\"n\", \"TI\"]}], \"\", RowBox[{\"2\", \"/\", StyleBox[\"n\",
    \"TI\"]}], \"\", StyleBox[\"[Ellipsis]\", \"TR\"], \"\", \"1\"}],
    \"}\"\}]\) . \!\(\*RowBox[{\"Subdivide\", \"[\", RowBox[{SubscriptBox[StyleBox[\"x\",
    \"TI\"], StyleBox[\"max\", \"TI\"]], \"\", StyleBox[\"n\", \"TI\"]}],
    \"]\"}]\) generates the list of values obtained by subdividing the interval 0
    to \!\(\*SubscriptBox[StyleBox[\"x\", \"TI\"], StyleBox[\"max\", \"TI\"]]\)
    into \!\(\*StyleBox[\"n\", \"TI\"]\) equal parts. \!\(\*RowBox[{\"Subdivide\",
    \"[\", RowBox[{SubscriptBox[StyleBox[\"x\", \"TI\"], StyleBox[\"min\", \"TI\"]],
    \"\", SubscriptBox[StyleBox[\"x\", \"TI\"], StyleBox[\"max\", \"TI\"]], \"\",
    StyleBox[\"n\", \"TI\"]}], \"]\"}]\) generates the list of values from subdividing
    the interval \!\(\*SubscriptBox[StyleBox[\"x\", \"TI\"], StyleBox[\"min\", \"TI\"]]\)
    to \!\(\*SubscriptBox[StyleBox[\"x\", \"TI\"], StyleBox[\"max\", \"TI\"]]\) .";

  Begin["`Private`"];

  Subdivide[xmin_, xmax_, n_] := xmin + (xmax - xmin) Range[0, n] / n;
  Subdivide[xmax_, n_] := Subdivide[0, xmax, n];
  Subdivide[n_] := Subdivide[0, 1, n];

  End[];
]
```

## FieldArrow

*In[ ]:=*

```
FieldArrow::usage =
  "FieldArrow[kind,f,{x,y,z}] produces a field arrow associated with the field function
  f at the position (x,y,z), where the kind can be \"major\" (for
  a major-axis arrow) or \"spin\" (for the electric spin vector).
  FieldArrow[kind,f,{x,y,z},A] multiplies the arrow length by an amplitude A.
  FieldArrow[kind,f,{x,y,z},A,zoom] magnifies the position by the specified zoom factor.
  FieldArrow[kind,f,{x,y,z},A,zoom,{tx,ty,tz}]
  translates the position of the arrow by an offset {tx,ty,tz}.";

FieldArrowheadFunction::usage =
  "FieldArrowheadFunction is an option for FieldArrow and PlotFieldArrows
  that specifies a function of the norm of the arrow vector
  that should output the size of arrowheads to use.";
FieldArrowThicknessFunction::usage = "FieldArrowThicknessFunction is an option
  for FieldArrow and PlotFieldArrows that specifies a function of the norm
  of the arrow vector that should output the tube diameter of the arrows.";

Begin["`Private`"];
Protect[FieldArrowThicknessFunction, FieldArrowheadFunction];
```

```

Options[FieldArrow] = {NormFunction -> Norm, PlotStyle -> {},
  FieldArrowThicknessFunction -> None, FieldArrowheadFunction -> None};

FieldArrow[kind_, fieldFunction_, {x_, y_, z_}, opts : OptionsPattern[]] :=
  FieldArrow[kind, fieldFunction, {x, y, z}, 1, 1, {0, 0, 0}, opts]
FieldArrow[kind_, fieldFunction_, {x_, y_, z_}, amplitude_, opts : OptionsPattern[]] :=
  FieldArrow[kind, fieldFunction, {x, y, z}, amplitude, 1, {0, 0, 0}, opts]
FieldArrow[kind_, fieldFunction_, {x_, y_, z_}, amplitude_, zoom_, opts : OptionsPattern[]] :=
  FieldArrow[kind, fieldFunction, {x, y, z}, amplitude, zoom, {0, 0, 0}, opts]

FieldArrow[kind_, fieldFunction_, {x_, y_, z_}, amplitude_,
  zoom_, translation_, opts : OptionsPattern[]] := Block[{vect},
  vect = Which[
    kind == "major", JonesMajorAxisA[fieldFunction[x, y, z], OptionValue[NormFunction]],
    kind == "minor", JonesMinorAxisB[fieldFunction[x, y, z], OptionValue[NormFunction]],
    kind == "spin", SpinE[fieldFunction[x, y, z]]
  ];
  {
    If[
      Not[OptionValue[FieldArrowheadFunction] === None],
      Arrowheads[OptionValue[FieldArrowheadFunction][amplitude Norm[vect]]],
      {}],

    ColorData[If[kind == "major", "SolarColors", "Rainbow"]][ $\frac{-\text{Re}[\text{vect}[[3]] + 1]}{2 \text{Norm}[\text{vect}]}$ ],

    OptionValue[PlotStyle],

    Arrow[Tube[
      Chop[{
        zoom {x, y, z} + translation,
        zoom {x, y, z} + translation + amplitude × vect
      }],
      If[
        Not[OptionValue[FieldArrowThicknessFunction] === None],
        OptionValue[FieldArrowThicknessFunction][amplitude Norm[vect]],
        ## &[]
      ]
    ]
  ]
}
End[];

```

## PlotFieldArrows



In[ ]:=

```

PlotFieldArrows::usage =
  "PlotFieldArrows[f,kind,rmax] plots a field-arrow plot (for arrows of the
    chosen kind, either \"major\" axis or \"spin\" vector) for the
    vector field function f, for radial coordinate from 0 to rmax.
  PlotFieldArrows[f,kind,sm,A] plots a field-arrow plot with amplitude A multiplying each arrow.
  PlotFieldArrows[f,kind,sm,A,zoom] plots a
    field-arrow plot with a magnification zoom on the arrow positions.
  PlotFieldArrows[f,kind,sm,A,zoom,{tx,ty,tz}] plots a field-arrow
    plot with an offset of {tx,ty,tz}.";

Begin["`Private`"];
Options[PlotFieldArrows] = {RadialPoints -> 10, FirstRingPoints -> 4, PlotStyle -> {},
  NormFunction -> Norm, FieldArrowThicknessFunction -> None, FieldArrowheadFunction -> None};

PlotFieldArrows[fieldFunction_, kind_, rmax_, opts : OptionsPattern[]] :=
  PlotFieldArrows[fieldFunction, kind, rmax, 1, 1, {0, 0, 0}, opts]
PlotFieldArrows[fieldFunction_, kind_, rmax_, amplitude_, opts : OptionsPattern[]] :=
  PlotFieldArrows[fieldFunction, kind, rmax, amplitude, 1, {0, 0, 0}, opts]
PlotFieldArrows[fieldFunction_, kind_, rmax_, amplitude_, zoom_, opts : OptionsPattern[]] :=
  PlotFieldArrows[fieldFunction, kind, rmax, amplitude, zoom, {0, 0, 0}, opts]

PlotFieldArrows[fieldFunction_, kind_, rmax_,
  amplitude_, zoom_, translation_, opts : OptionsPattern[]] := Block[{},
Graphics3D[{
  Arrowheads[0.02],
  Thickness[0.005],
  Table[
    FieldArrow[kind, fieldFunction, Join[point, {0}], amplitude,
      zoom, translation, Sequence @@ FilterRules[{opts}, Options[FieldArrow]]]
    , {point, UniformRadialGrid[OptionValue[RadialPoints],
      OptionValue[FirstRingPoints], rmax/OptionValue[RadialPoints]]}]
  }]]
]

End[];

```

## FieldEllipse

In[ ]:=

```
FieldEllipse::usage =
  "FieldEllipse[f,{x,y,z},a] plots the polarization ellipse of the vector field
  function f at position (x,y,z), with amplitude multiplier a.
  FieldEllipse[f,{x,y,z},a,normFunction] normalizes the field values by
  the maximum of normFunction over a period. The default is
  (1&) and is inactive; for unit normalization, set to Norm.
  FieldEllipse[f,{x,y,z},a,normFunction,zoom] magnifies the position by the specified zoom factor.
  FieldEllipse[f,{x,y,z},a,normFunction,zoom,{tx,ty,tz}]
  translates the position of the ellipse by an offset {tx,ty,tz}.";

Begin["`Private`"];
Options[FieldEllipse] = {PlotPoints -> 72 + 1};

FieldEllipse[fieldFunction_, {x_, y_, z_}, amplitude_, normFunction_ : (1 &), zoom_ : 1,
  translation_ : {0, 0, 0}, opts : OptionsPattern[] ] := Block[{field, points, norm},
  field = fieldFunction[x, y, z];
  points = Table[ Re[ $e^{-i \omega t}$  field], { $\omega t$ , 0., 360°,  $\frac{360^\circ}{\text{OptionValue[PlotPoints]} - 1}$  }];
  norm = Max[normFunction /@ points];
  {
    Polygon[
      Map[Function[translation + zoom {x, y, 0} +  $\frac{\text{amplitude}}{\text{norm}}$  #], points]
    ]
  }
];

End[];
```

## PlotFieldEllipses

In[ ]:=

```
PlotFieldEllipses::usage =
  "PlotFieldEllipses[f,amplitude,rmax] plots a field-ellipse plot for the
    vector field function f, for radial coordinate from 0 to rmax.
PlotFieldEllipses[f,amplitude,rmax,normFunction] uses the given
    normFunction to normalize the ellipses.
PlotFieldEllipses[f,amplitude,rmax,normFunction,zoom] uses a
    magnification zoom on the ellipse positions.
PlotFieldEllipses[f,amplitude,rmax,normFunction,zoom,{tx,ty,tz}]
    translates the ellipses by an offset {tx,ty,tz}.";

RadialPoints::usage = "RadialPoints is an option for PlotFieldArrows
  and PlotFieldEllipses that indicates how many radial points to use.";
FirstRingPoints::usage = "FirstRingPoints is an option for PlotFieldArrows and
  PlotFieldEllipses that indicates how many points to use on the first ring.";

Begin["`Private`"];
Protect[RadialPoints];
Options[PlotFieldEllipses] =
  {PlotStyle -> {}, PlotPoints -> 72 + 1, RadialPoints -> 10, FirstRingPoints -> 4};

PlotFieldEllipses[fieldFunction_, amplitude_, rmax_, opts : OptionsPattern[]] :=
  PlotFieldEllipses[fieldFunction, amplitude, rmax, Norm, 1, {0, 0, 0}, opts]
PlotFieldEllipses[fieldFunction_, amplitude_, rmax_, normFunction_, opts : OptionsPattern[]] :=
  PlotFieldEllipses[fieldFunction, amplitude, rmax, normFunction, 1, {0, 0, 0}, opts]
PlotFieldEllipses[fieldFunction_, amplitude_, rmax_,
  normFunction_, zoom_, opts : OptionsPattern[]] :=
  PlotFieldEllipses[fieldFunction, amplitude, rmax, normFunction, zoom, {0, 0, 0}, opts]

PlotFieldEllipses[fieldFunction_, amplitude_, rmax_,
  normFunction_, zoom_, translation_, opts : OptionsPattern[]] := Block[{},
Graphics3D[{
  EdgeForm[{Thick, Black}],
  FaceForm[{Blue, Specularity[0]}],
  OptionValue[PlotStyle],
  Table[
    FieldEllipse[fieldFunction, Join[point, {0}], amplitude, normFunction,
      zoom, translation, Sequence @@ FilterRules[{opts}, Options[FieldEllipse]]],
    {point, UniformRadialGrid[OptionValue[RadialPoints],
      OptionValue[FirstRingPoints], rmax/OptionValue[RadialPoints]]}]
  }]]
]

End[];
```

# Benchmarking

## Verifying the solutions

### The analytical expressions

**ComplexFocusHelicityE[1, 1, 1, {x, y, z}, q]**

$$\begin{aligned} & \left\{ \sqrt{3} \pi (x + i y) \left( (-3 i - i q + z) \text{AnalyticalBesselJ}[2, \{x, y, -i q + z\}] + \right. \right. \\ & \quad \left. \left. (-i q^2 + x y + 2 q z + i (y^2 + z^2)) \text{AnalyticalBesselJ}[3, \{x, y, -i q + z\}] \right), \right. \\ & - \sqrt{3} \pi (x + i y) \left( -(3 + q + i z) \text{AnalyticalBesselJ}[2, \{x, y, -i q + z\}] + \right. \\ & \quad \left. (-q^2 + x^2 + i x y - 2 i q z + z^2) \text{AnalyticalBesselJ}[3, \{x, y, -i q + z\}] \right), - \sqrt{3} \pi (x + i y)^2 \\ & \quad \left. \left( \text{AnalyticalBesselJ}[2, \{x, y, -i q + z\}] + (q + i z) \text{AnalyticalBesselJ}[3, \{x, y, -i q + z\}] \right) \right\} \end{aligned}$$

### Verifying the defining properties ( $\nabla \cdot E = 0$ , $(\nabla^2 + k^2)E = 0$ ) via symbolic computation

Divergence:

```
Block[{l = 0, m = 0},
  FullSimplify[
    Div[
      ComplexFocusHelicityE[l, m, 1, k {x, y, z}, q]
    , {x, y, z}]
  ]
]
```

Helmholtz equation:

```
Block[{l = 0, m = 0},
  FullSimplify[
    ReplaceAll[
      Plus[
        Laplacian[
          ComplexFocusHelicityE[l, m, 1, k {x, y, z}, q]
        , {x, y, z}],
        k^2 ComplexFocusHelicityE[l, m, 1, k {x, y, z}, q]
      ],
      {AnalyticalBesselJ[m_, {x_, y_, z_}] =>  $\frac{\text{SphericalBesselJ}[\text{Abs}[m], \sqrt{x^2 + y^2 + z^2}]}{(x^2 + y^2 + z^2)^{\text{Abs}[m]/2}}$ 
    ]
  ]
]
```

## Verifying the defining properties directly

```
Block[{q = 5, laplacian, divergence},
  Flatten[
    Table[Table[
      Join[
        Table[
          laplacian[x_, y_, z_] = FullSimplify[
            Laplacian[ComplexFocusHelicityE[l, m, 1, {x, y, z}, q][[j]], {x, y, z}]
          ];
          Plot[{
            Re[laplacian[x, 0, 0]], Re[-ComplexFocusHelicityE[l, m, 1, {x, 0, 0}, q][[j]],
            Im[laplacian[x, 0, 0]], Im[-ComplexFocusHelicityE[l, m, 1, {x, 0, 0}, q][[j]]
          }, {x, 0, 10}
          , ImageSize → 250
          , Frame → True
          , PlotLabel → Row[{"l=", l, "m=", m, " ", {"x", "y", "z"}[[j]]}
          , PlotRange → Full
        ]
        , {j, 1, 3}], {
          divergence[x_, y_, z_] = FullSimplify[
            Div[ComplexFocusHelicityE[l, m, 1, {x, y, z}, q], {x, y, z}]
          ];
          Plot[{
            Re[divergence[x, 0, 0]], Im[divergence[x, 0, 0]]
          }, {x, 0, 10}
          , ImageSize → 250
          , Frame → True
          , PlotLabel → Row[{"l=", l, "m=", m, " div"}]
          , PlotRange → Full
        ]
      }]]
    , {m, 0, 1}], {1, 0, 1}]
  , 1]
]
```

## Intensity plots

```
Block[{R = 3 q, Δ = 0.5, F0 = 0.075, g, l = 1, m = 1, s = 1, k = 1, q = 30},
  g[x_, y_, z_] := ComplexFocusHelicityE[l, m, s, k {x, y, z}, k q];

  ContourPlot[
    Norm[g[x, 0, z]]2
    , {z, -R, R}, {x, -R, R}
    , PlotRangePadding → None
    , ImageSize → 650
    , PlotPoints → 35
    , PlotRange → Full
    , Contours → 15
  ]
]
```

## Polarization ellipses

```

In[ ]:= DynamicModule[{viewPoint = {0, 0, 2}, viewVertical = {0, 1, 0}},
  Block[{R = 3  $\sqrt{q}$ ,  $\Delta = 0.5 \sqrt{q}$ , F0 = 0.2  $\sqrt{q}$ , g0, g, l = 0, m = 0, s = 1},
    SlideView[
      Table[
        g0 = Norm[ComplexFocusHelicityE[l, m, s, {0, 0, 0}, q]];
        g[x_, y_, z_] :=  $\frac{1}{g0}$  ComplexFocusHelicityE[l, m, s, {x, y, z}, q];

        Show[{
          PlotFieldEllipses[g, F0, R, 1 &, FirstRingPoints → 7],
          PlotFieldArrows[g, "major", R, F0, 1, {0, 0, 0}
            , FirstRingPoints → 7
            , NormFunction → Function[1]
            , PlotStyle → Directive[Red]
            , FieldArrowheadFunction → Function[Min[{0.001 F0,  $\frac{F0}{25}$  #}]]
            , FieldArrowThicknessFunction → Function[0.075 F0]
          ],
          PlotFieldArrows[g, "minor", R, F0, 1, {0, 0, 0}
            , FirstRingPoints → 7
            , NormFunction → Function[1]
            , PlotStyle → Directive[Blue]
            , FieldArrowheadFunction → Function[Min[{0.001 F0,  $\frac{F0}{25}$  #}]]
            , FieldArrowThicknessFunction → Function[0.075 F0]
          ]
        }
        , Axes → True
        , ImageSize → 650
        , SphericalRegion → True
        , Lighting → "Neutral"
        , ViewPoint → Dynamic[viewPoint]
        , ViewVertical → Dynamic[viewVertical]
        , ViewVertical → {0, 0, 1}
        , PlotLabel → q
        , PlotRange → {1.1 R {-1, 1}, 1.1 R {-1, 1},  $\Delta$  {-1, 1}}
      ]
    , {q, PowerRange[1, 100, 100.2]}]]]

```


## With varying wavenumber $k$

When called as `ComplexFocusHelicityE[l,m,s,k {x,y,z},q]`, the `ComplexFocus` functions produce essentially the same plot, but at different wavelengths, so the plot is identical once the chosen plot points (i.e. `Range[-R,R, $\Delta$ ]`) are scaled appropriately.

However, it is important to keep tabs on the meaning of  $q$ :

- If used as ComplexFocusHelicityE[l,m,s,k {x,y,z},q] then the Rayleigh range is  $q/k$ .
- If used as ComplexFocusHelicityE[l,m,s,k {x,y,z},k q] then the Rayleigh range is  $q$ .

```
DateString[]
DynamicModule[{viewPoint = {1.5, 0, 0.2}, viewVertical = {0, 0, 1}},
  SlideView[Table[
    Block[{R = 3  $\sqrt{q}$  / k,  $\Delta$  = 0.5  $\sqrt{q}$  / k, F0 = 0.3  $\sqrt{q}$  / k, g0, g, l = 0, m = 0, s = 1, q = 2},
      g0 = Norm[ComplexFocusHelicityE[l, m, s, {0, 0, 0}, q]];
      g[x_, y_, z_] :=  $\frac{1}{g0}$  ComplexFocusHelicityE[l, m, s, k {x, y, z}, q];

    Show[{
      Graphics3D[{
        Table[{
          {Red, Blue}[[j]],
          Arrowheads[Min[{0.0125,  $\frac{1}{25}$  Norm[F0 {Re, Im}[[j]][#] & [g[x, y, 0]]]}]],
          Arrow[Tube[{
            {x, y, 0},
            {x, y, 0} + F0 {Re, Im}[[j]][#] & [g[x, y, 0]]
          }]],
        }, {j, 1, 2}
      ], {x, -R, R,  $\Delta$ }, {y, -R, R,  $\Delta$ }
    ]],
    Table[
      Graphics3D[{
        EdgeForm[{Thick, Black}],
        FaceForm[{, Specularity[0]}],
        Polygon[Table[
          {x, y, 0} + F0 Re[ $e^{-i \omega t}$  g[x, y, 0]]
        ], { $\omega t$ , 0., 360°, 5°}]]
      ], {x, -R, R,  $\Delta$ }, {y, -R, R,  $\Delta$ }
    ]
  ], Axes → True
  , ImageSize → 950
  (*, SphericalRegion → True*)
  , Lighting → "Neutral"
  , Ticks → {Automatic, Automatic, None}
  , ViewPoint → Dynamic[viewPoint]
  , ViewVertical → Dynamic[viewVertical]
  , PlotLabel → {q, k}
  , PlotRange → {1.05 {-R, R}, 1.05 {-R, R}, 0.05 R {-1, 1}}
  , BoxRatios → Automatic
]
], {k, 1, 2}]]]
DateString[]
Mon 9 Nov 2020 15:20:54
Mon 9 Nov 2020 15:21:14
```

```

In[ ]:=DateString[]
DynamicModule[{viewPoint = {1.5, 0, 0.2}, viewVertical = {0, 0, 1}},
  SlideView[Table[
    Block[{R = 3  $\sqrt{q}$  / k,  $\Delta = 0.5 \sqrt{q}$  / k, F0 = 0.3  $\sqrt{q}$  / k, g0, g, l = 0, m = 0, s = 1, q = 2},
      g0 = Norm[ComplexFocusHelicityE[l, m, s, {0, 0, 0}, q]];
      g[x_, y_, z_] :=  $\frac{1}{g0}$  ComplexFocusHelicityE[l, m, s, k {x, y, z}, q];

    Show[{
      PlotFieldEllipses[g, F0, R, 1 &, FirstRingPoints → 7],
      PlotFieldArrows[g, "major", R, F0, 1, {0, 0, 0}
        , FirstRingPoints → 7
        , NormFunction → Function[1]
        , PlotStyle → Directive[Red]
        , FieldArrowheadFunction → Function[Min[{0.001 F0,  $\frac{F0}{25}$  #}]]
        , FieldArrowThicknessFunction → Function[0.075 F0]
      ],
      PlotFieldArrows[g, "minor", R, F0, 1, {0, 0, 0}
        , FirstRingPoints → 7
        , NormFunction → Function[1]
        , PlotStyle → Directive[Blue]
        , FieldArrowheadFunction → Function[Min[{0.001 F0,  $\frac{F0}{25}$  #}]]
        , FieldArrowThicknessFunction → Function[0.075 F0]
      ]
    ]
  ], Axes → True
  , ImageSize → 650
  , SphericalRegion → True
  , Lighting → "Neutral"
  , ViewPoint → Dynamic[viewPoint]
  , ViewVertical → Dynamic[viewVertical]
  , ViewVertical → {0, 0, 1}
  , PlotLabel → q
  , PlotRange → {1.1 R {-1, 1}, 1.1 R {-1, 1},  $\Delta$  {-1, 1}}
  ]
  , {k, 1, 2}]]
]
DateString[]

```

## Package closure

### End of package

```

In[ ]:= EndPackage[];

```



## Add to distributed contexts

In[ ]:=

```
DistributeDefinitions["ComplexFocus`"];
```