ARMSupport

This file provides supporting functions for Analytical *R*-Matrix calculations.

Version history

1.0.0	Initial set-up – pulling functions from a nebulous cloud of files.
1.0.1	- Made error handling on coulombCorrection more flexible by allowing for customized ReportingFunction. - Fixed options handling on coulombCorrection (gave an error if no option was given).
1.0.2	Added v2Tolerance as an option for closestApproachTimesPath.
1.0.3	Fixed classicalClosestApproach - it gave wrong times for some reason.
1.0.4	Added new exception to path chooser - turning point on $-\pi/2 < \mathrm{Re}(\omega t) < \pi/2$ to help avoid $\mathrm{Re}\left(\boldsymbol{r}_{\mathrm{cl}}(t)^2\right) < 0$ regions.
1.0.5	Added rInit support to trajectory functions.
1.0.6	Added non-standard t_s support to trajectory functions (i.e. forcets).
1.0.7	Added package export functionality.
1.0.8	Sealed off implementations inside \`Private\` contexts.
1.0.9	Changed order of documentation and \`Private\` sections to ensure public symbols do not end up as \`Private\`.
1.0.10	SetSharedFunction calls set to evaluate only on master kernel evaluation.
1.0.11	Fixes to makeTCAsFromRange, on contexts and options issues.
1.0.12	Admin changes. Changed file name, added version number and timestamp functions pulled in from RB-SFA package.
1.0.13	Pulled in SFT implementation from notes.

Export features

To Quantum Optics Dynamic Dashboard

```
Block[{directory},
  directory =
   Which[$MachineName == "ph-dtc11-09", "D:\\Work\\CQD\\Project\\Code\\QuODD\\",
    True, "~/Work/CQD/Project/Code/QuODD/"];
  Column[{
    CopyFile[NotebookFileName[],
     directory <> "ARMSupport.nb", OverwriteTarget → True],
    CopyFile[StringReplace[NotebookFileName[], ".nb" → ".m"],
     directory <> "ARMSupport.m", OverwriteTarget → True]
   }]];
```

Version number

```
$ARMSupportVersion::usage =
  "$ARMSupportVersion prints the current version of the ARMSupport
    package in use and its timestamp.";
$ARMSupportTimestamp::usage = "$ARMSupportTimestamp prints the
    timestamp of the current version of the ARMSupport package.";
Begin["`Private`"];
$ARMSupportVersion := "ARMSupport v1.0.13, " <> $ARMSupportTimestamp;
End[];
```

The timestamp is updated every time the 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[{"Begin[\"`Private`\"];\n$ARMSupportTimestamp=\"" <>
              DateString[] <> "\";\nEnd[];"}]]
         , "Input", InitializationCell → True, CellTags → "version-timestamp"
        ], None, AutoScroll → False];
       NotebookSave[]
      ), PassEventsDown → True}
 ];
```

To reset this behaviour to normal, evaluatethe cell below

Timestamp

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

Begin["`Private`"];
$ARMSupportTimestamp = "Mon 30 May 2016 22:20:13";
End[];
```

Git commit hash and message

```
$ARMSupportDirectory::usage =
  "$ARMSupportDirectory is the directory where the current
    RB-SFA package instance is located.";
$ARMSupportCommit::usage = "$ARMSupportCommit returns the git commit
    log at the location of the RB-SFA package if there is one.";
$ARMSupportCommit::OS = "$ARMSupportCommit has only been tested on Linux.";
```

```
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.*)
  $ARMSupportDirectory = StringReplace[
    DirectoryName[softLinkTestString[[2]]], {" " → "\\ "}],
  $ARMSupportDirectory = StringReplace[
     DirectoryName[$InputFileName], {" " → "\\ "}];
 ]]
$ARMSupportCommit :=
  (If[$OperatingSystem ≠ "Unix", Message[$ARMSupportCommit::OS]];
   StringJoin[Riffle[ReadList[
      "!cd " <> $ARMSupportDirectory <> " && git log -1", String], {"\n"}]]);
End[];
```

General functions

Sundry initialization

```
BeginPackage["ARMSupport`", {"EPToolbox`"}];
```

```
stdpars = {0.05, 0.055, 1.007};
```

```
r0 = {0, 0, 0};
```

This needs to be run before any parallel evaluation:

```
ParallelEvaluate[Needs["EPToolbox`",
   "/home/episanty/Work/CQD/Project/Code/EPToolbox/EPToolbox/EPToolbox.m"]];
```

t_s and relatives

```
ts::usage = "ts[{po, py, pp}, {F, ω, κ}] Returns the saddle point tₛ directly.

ts[pp, κ, ω, F, po, py] Returns the saddle point tₛ directly.";
t0::usage = "t0[{po, py, pp}, {F, ω, κ}] Returns t₀=Re[tₛ] directly.

t0[pp, κ, ω, F, po, py] Returns t₀=Re[tₛ] directly.";
τ::usage = "τ[{po, py, pp}, {F, ω, κ}] Returns τ_T=Im[tₛ] directly.

τ[pp, κ, ω, F, po, py] Returns τ_T=Im[tₛ] directly.";
tκ::usage =
  "tκ[{po, py, pp}, {F, ω, κ}] Returns the starting point t_κ directly.

tκ[pp, κ, ω, F, po, py] Returns the starting point t_κ directly.";
```

```
Begin["`Private`"];
ts[pp_, κ_, ω_, F_, po_, py_] := (1/ω) ArcSin[(ω/F) pp + ⅈ (ω/F) Sqrt[κ^2 + po^2 + py^2]]
ts[{po_, py_, pp_}, {F_, ω_, κ_}] := ts[pp, κ, ω, F, po, py]

t0[{po_, py_, pp_}, {F_, ω_, κ_}] := Re[ts[pp, κ, ω, F, po, py]]
t0[pp_, κ_, ω_, F_, po_, py_] := Re[ts[pp, κ, ω, F, po, py]]

τ[{po_, py_, pp_}, {F_, ω_, κ_}] := Im[ts[pp, κ, ω, F, po, py]]
τ[pp_, κ_, ω_, F_, po_, py_] := Im[ts[pp, κ, ω, F, po, py]]

tκ[pp_, κ_, ω_, F_, po_, py_] := ts[pp, κ, ω, F, po, py] - ⅈ / κ^2
tκ[{po_, py_, pp_}, {F_, ω_, κ_}] := tκ[pp, κ, ω, F, po, py]
End[];
```

Transition times and momenta

Classical transition times are those times t_r for which a classical orbit has a recollision at zero velocity: $\boldsymbol{v}(t_r) = 0$ and $\mathrm{Re}\left[\int_{t_\kappa}^{t_r} \boldsymbol{p} + \boldsymbol{A}(\tau)\, d\tau\right] = 0$.

Classical transitions times and momenta

getClassical Transition returns exact numerical solutions.

getClassicalTransition

```
getClassicalTransition::usage =
  "getClassicalTransition[n, {F, ω, κ}] Returns pz and
    tr in atomic units as a list of replacement rules.
getClassicalTransition[range, {F, ω, κ}]
getClassicalTransition[n, F, ω, κ]";

pz::usage = "pz represents the z component of momentum."
tr::usage = "tr represents the return
   time as calculated by getClassicalTransition and friends."
pz represents the z component of momentum.

tr represents the return time as calculated by getClassicalTransition and friends.
```

```
Begin["`Private`"];
getClassicalTransition[n_, {F_, ω_, κ_}] := getClassicalTransition[n, F, ω, κ]
getClassicalTransition[range_List, {F_, ω_, κ_}] :=
 (getClassicalTransition[#, {F, ω, κ}] & /@ range)
getClassicalTransition[n_, F_, ω_, κ_] := Module[{getTimes, t00, zinit},
  getTimes[pz_] := ({t0 → Re[#], τ → Im[#]} & [1/ω ArcSin[ω/F (pz + ⅈ κ)]]);
  t00[pz_?NumericQ] := (t0 /. getTimes[pz]);
  zinit[pz_?NumericQ] :=
   Re[F/ω^2 (Cos[ω t0] - Cos[ω (t0 + ⅈ τ - ⅈ/κ^2)]) /. getTimes[pz]];
  FindRoot[
   {pz (tr - t00[pz]) + F/ω^2 (Cos[ω tr] - Cos[ω t00[pz]]) + zinit[pz] == 0,
    pz - F/ω Sin[ω tr] == 0}
   , {{pz, 0}, {tr, (n + 1) π/ω}}
  ]
 ]
End[];
```

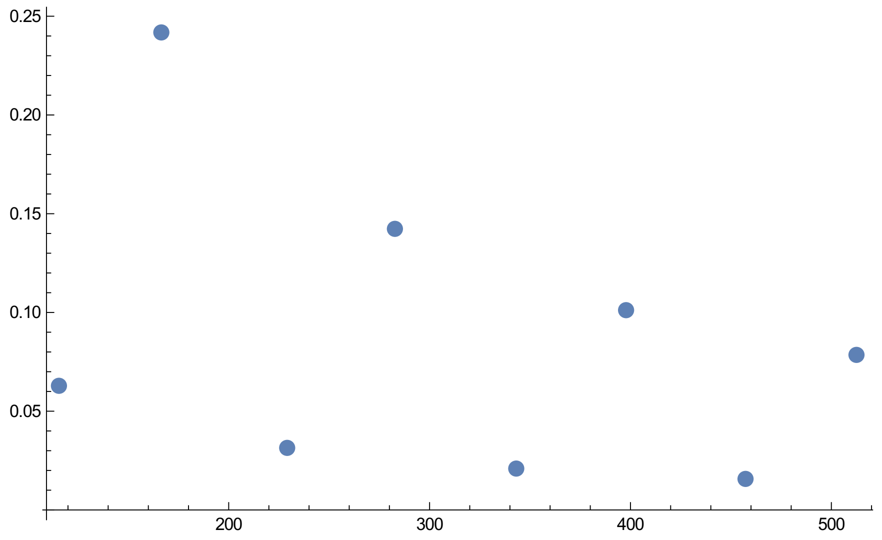
```
getClassicalTransition[Range[8], stdpars]
```

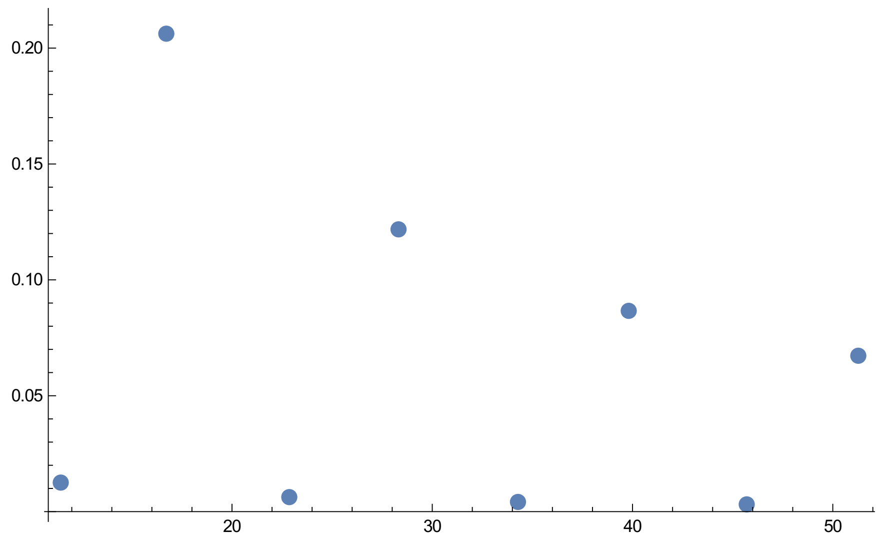
```
{{pz → 0.062865, tr → 115.498}, {pz → 0.241791, tr → 166.465},
 {pz → 0.0314275, tr → 229.108}, {pz → 0.142337, tr → 282.741},
 {pz → 0.0209511, tr → 343.138}, {pz → 0.101158, tr → 397.812},
 {pz → 0.0157131, tr → 457.273}, {pz → 0.0785167, tr → 512.507}}
```

Benchmarking

Momentum against time.

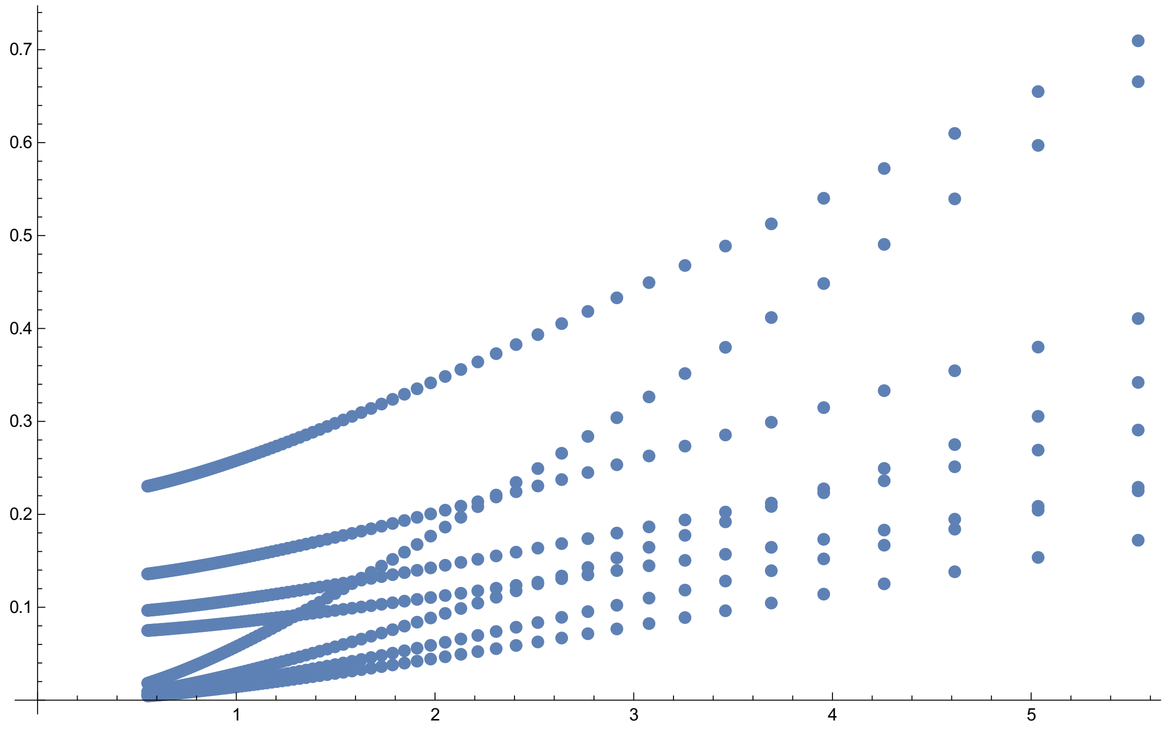
```
Row[{
  ListPlot[{tr, pz} /. getClassicalTransition[Range[8], {0.05, 0.055, 1.007}],
   ImageSize → 450],
  ListPlot[{tr, pz} /. getClassicalTransition[Range[8],
     {10 × 0.05, 10 × 0.055, 1.007}], ImageSize → 450]
 }]
```





Normalized momentum against γ.

```
ListPlot[
 Flatten[Table[
   Block[{ω = 0.055, κ = 1.007},
    {ω κ/F, ω pz/F} /. getClassicalTransition[Range[8], {F, ω, κ}]
   ]
   , {F, 0.01, 0.1, 0.001}], 1]
 , ImageSize → 600
]
```



Sandbox

You’re probably here to calculate classical transitions for some given parameters. Have a go.

```
pz /. getClassicalTransition[Range[4], {0.05, 45.6/3100, 1.07}]
```

```
{0.0240298, 0.755595, 0.0120147, 0.446211}
```

```
27.2 pz^2/2 /. getClassicalTransition[Range[4], {0.05, 45.6/3100, 1.07}]
```

```
{0.00785306, 7.76456, 0.0019632, 2.70781}
```

Linearized solutions.

These functions return classical transition times which have been linearized to first order in ***p***, i.e.

$p_z = \frac{F}{\omega}\frac{(-1)^n+\sqrt{1+\gamma^2}}{(n+1)\,\pi}$. The reduced results return $\frac{\omega}{F}\boldsymbol{p}$.

getLinearizedTransition

```
getLinearizedTransition::usage =
  "getLinearizedTransition[n, {F, ω, κ}] Returns pz and
    tr in atomic units as a list of replacement rules.
getLinearizedTransition[range, {F, ω, κ}]
getLinearizedTransition[n, F, ω, κ]";
```

```
Begin["`Private`"];
getLinearizedTransition[n_, {F_, ω_, κ_}] := getClassicalTransition[n, F, ω, κ]
getLinearizedTransition[range_List, {F_, ω_, κ_}] :=
 (getClassicalTransition[#, {F, ω, κ}] & /@ range)
getLinearizedTransition[n_, F_, ω_, κ_] :=
 {pz → F/ω getReducedLinearizedTransition[n, ω κ/F],
  tr → 1/ω ((n + 1) π + ArcSin[getReducedLinearizedTransition[n, ω κ/F]])}
End[];
```

getReducedLinearizedTransition

```
getReducedLinearizedTransition::usage =
  "getReducedLinearizedTransition[n, {F, ω, κ}] Returns ωpz/F directly.
getReducedLinearizedTransition[range, {F, ω, κ}]
getReducedLinearizedTransition[n, F, ω, κ]";
```

```
Begin["`Private`"];
getReducedLinearizedTransition[range_List, γ_] :=
 (getReducedLinearizedTransition[#, γ] & /@ range)
getReducedLinearizedTransition[n_, γ_] := ((-1)^n + Sqrt[1 + γ^2]) / (π (n + 1))
End[];
```

Full transitions. getFullLinearizedTransition and getFullReducedLinearizedTransition.

i.e. without neglecting terms in ω/κ^2.

```
getFullLinearizedTransition::usage =
  "getFullLinearizedTransition[n, {F, ω, κ}] Returns
    pz and tr in atomic units as a list of replacement
    rules, for the linearized case without neglecting ω/κ^2.
getFullLinearizedTransition[range, {F, ω, κ}]
getFullLinearizedTransition[n, F, ω, κ]";
getFullReducedLinearizedTransition::usage =
  "getFullReducedLinearizedTransition[n, {F, ω, κ}] Returns ωpz/F directly.
getFullReducedLinearizedTransition[range, {F, ω, κ}]
getFullReducedLinearizedTransition[n, F, ω, κ]";

Begin["`Private`"];
getFullLinearizedTransition[n_, {F_, ω_, κ_}] :=
 getFullLinearizedTransition[n, F, ω, κ]
getFullLinearizedTransition[range_List, {F_, ω_, κ_}] :=
 (getFullLinearizedTransition[#, {F, ω, κ}] & /@ range)
getFullLinearizedTransition[n_, F_, ω_, κ_] :=
```

$$\left\{pz \to \frac{F}{\omega}\,\text{getFullReducedLinearizedTransition}\left[n, \frac{\omega\,\kappa}{F}\right],\right.$$
$$\left. tr \to \frac{1}{\omega}\left((n+1)\,\pi + \text{ArcSin}\left[\text{getReducedLinearizedTransition}\left[n, \frac{\omega\,\kappa}{F}\right]\right]\right)\right\}$$

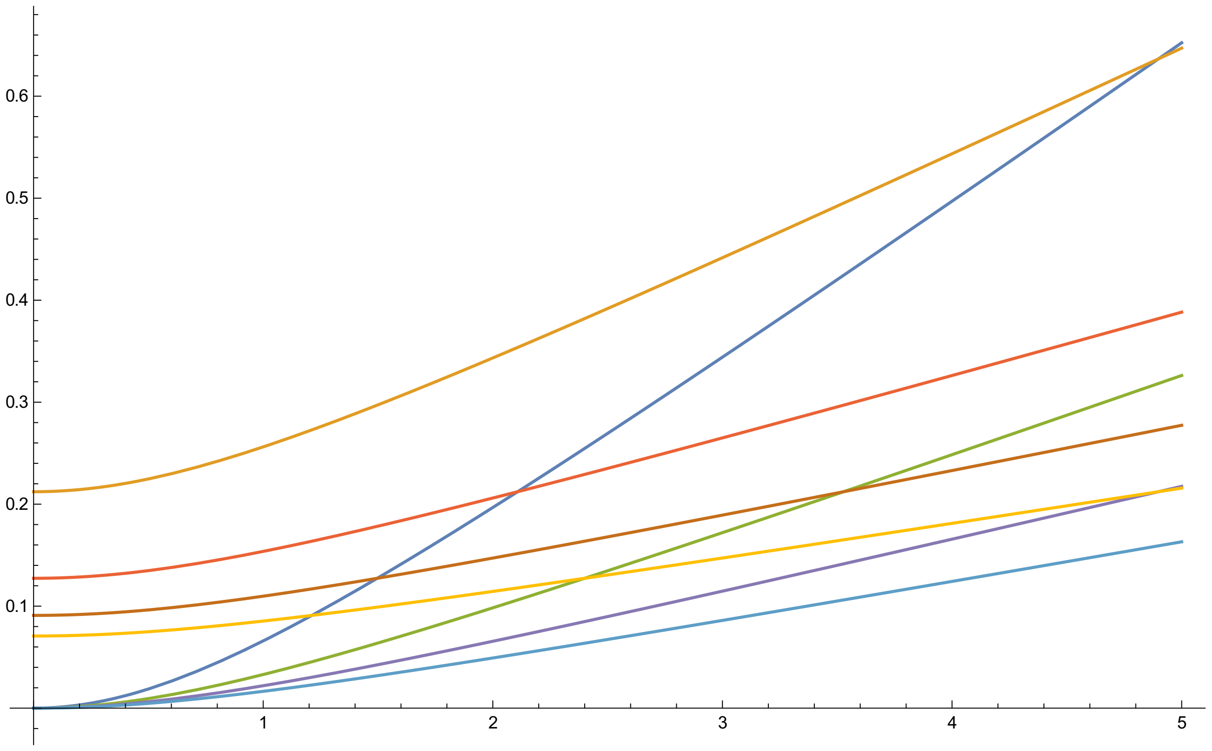
```
getFullReducedLinearizedTransition[range_List, F_, ω_, κ_] :=
 (getFullReducedLinearizedTransition[#, F, ω, κ] & /@ range)
getFullReducedLinearizedTransition[n_, F_, ω_, κ_] :=
```

$$\frac{(-1)^n + \sqrt{1 + \left(\frac{\omega\,\kappa}{F}\right)^2 \,\text{Cosh}\left[\frac{\omega}{\kappa^2}\right] - \frac{\omega\,\kappa}{F}\,\text{Sinh}\left[\frac{\omega}{\kappa^2}\right]}}{(n+1)\,\pi}$$

```
End[];
```

Benchmarking

```
Plot[Evaluate@getReducedLinearizedTransition[Range[8], γ], {γ, 0, 5}]
```



Complex-momentum solutions.

Complex-momentum solutions return complex times and momenta which are solutions to the complex equations $\boldsymbol{v}(t_r) = 0$ and $\int_{t_\kappa}^{t_r} \boldsymbol{p} + \boldsymbol{A}(\tau)\, d\tau = 0$.

getComplexTransition

```
getComplexTransition::usage =
  "getComplexTransition[n, {F, ω, κ}] Returns pz and tr
    in atomic units as a list of replacement rules.
getComplexTransition[range, {F, ω, κ}]
getComplexTransition[n, F, ω, κ]";
```

```
Begin["`Private`"];
getComplexTransition[n_, {F_, ω_, κ_}] := getComplexTransition[n, F, ω, κ]
getComplexTransition[range_List, {F_, ω_, κ_}] :=
 (getComplexTransition[#, {F, ω, κ}] & /@ range)
getComplexTransition[n_, F_, ω_, κ_] := Module[{tκκ},
  tκκ[pz_] := ts[pz, κ, ω, F, 0, 0] - i/κ^2;
  FindRoot[
   (ω pz/F) ((n + 1) π + ArcSin[ω pz/F] - ω tκκ[pz]) + (-1)^(n+1) Sqrt[1 - (ω pz/F)^2] - Cos[ω tκκ[pz]] == 0
   , {pz, 0.0}]
 ]
End[];
```

```
getComplexTransition[Range[2], stdpars]
```

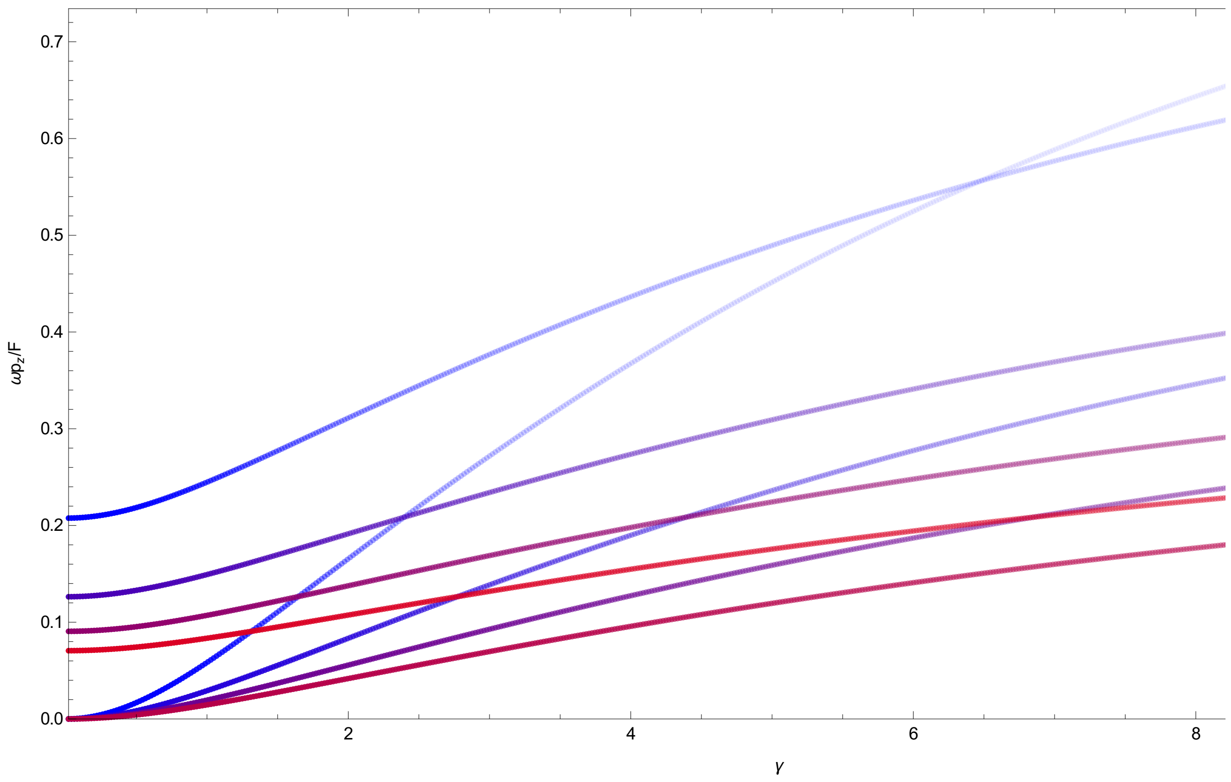
```
{{pz → 0.0627928 + 0.00212574 i}, {pz → 0.228599 + 0.00501516 i}}
```

Benchmarking

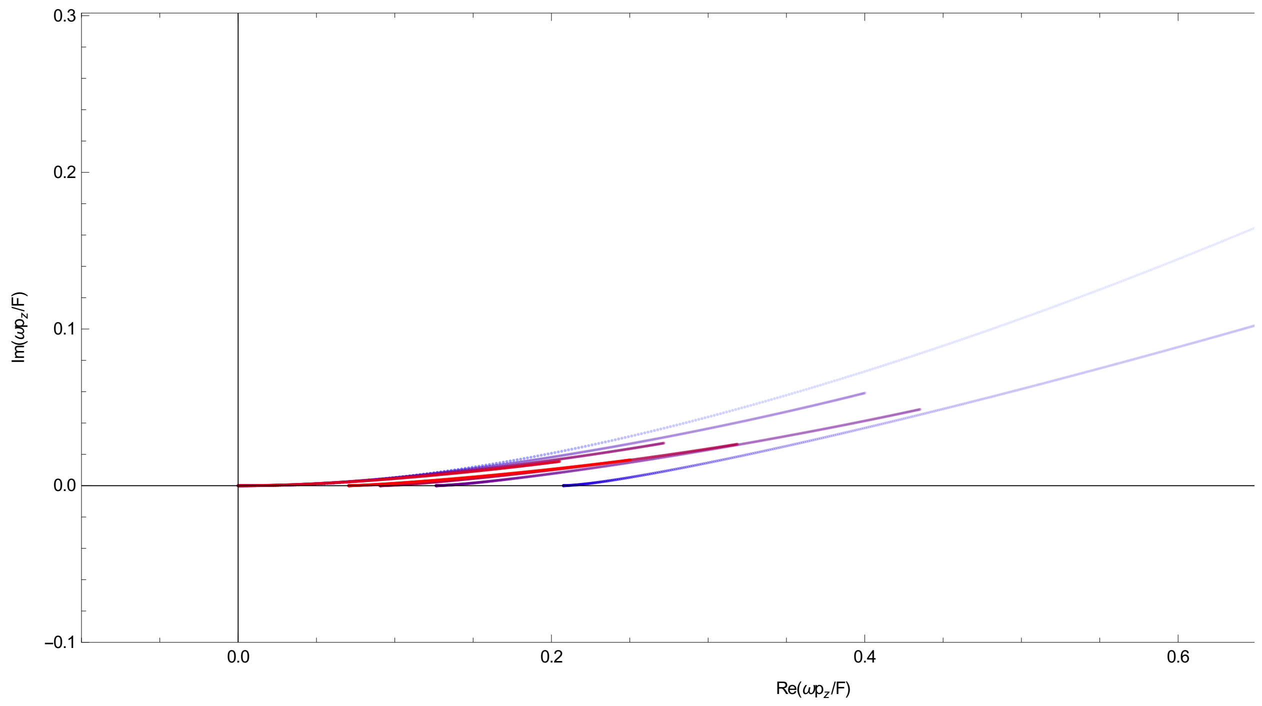
```
AbsoluteTiming[
 complexTransitionsBenchmarkingData = Flatten[Table[
     Block[{F = 0.05, κ = 1.007},
      {"k" → k, "γ" → ω κ/F, "ωpF" → ω pz/F} /. getComplexTransition[k, {F, ω, κ}]
     ]
     , {k, 1, 8}, {ω, 0.001, 0.5, 0.001}], 1];
]
```

```
{3.25438, Null}
```

```
Show[
 Graphics[
  {Quiet[Blend[{Blue, Red}, ("k" - 2)/7]], Opacity[1 / (1 + Im["ωpF"] / 0.01)],
    Point[{"γ", Re["ωpF"]}]} /. complexTransitionsBenchmarkingData
 ]
 , FrameLabel → {"γ", "ωp_z/F"}, PlotRangePadding → None,
 Frame → True, ImageSize → {800, 400}, AspectRatio → 0.5
]
```



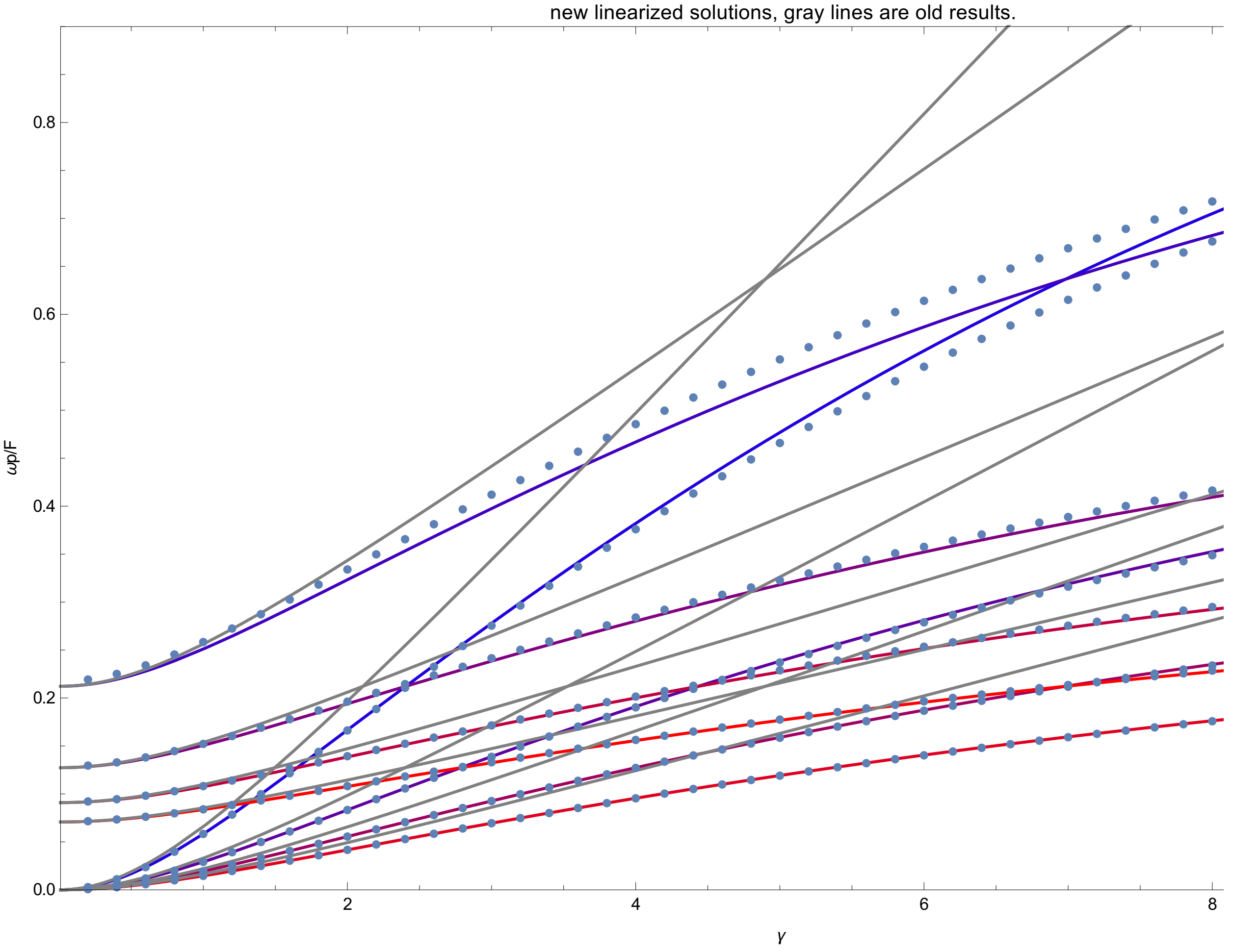
```
Graphics[
 {Quiet[Blend[{Blue, Red}, ("k" - 1)/7]], Opacity[1/(1 + Im["ωpF"] / 0.01)], PointSize[0.002],
   Point[{Re["ωpF"], Im["ωpF"]}]} /. complexTransitionsBenchmarkingData
 , PlotRangePadding → 0.1, Frame → True, Axes → {True, True}, AxesOrigin → {0, 0},
 FrameLabel → {"Re(ωp_z/F)", "Im(ωp_z/F)"}, ImageSize → 800
]
```



Comparison

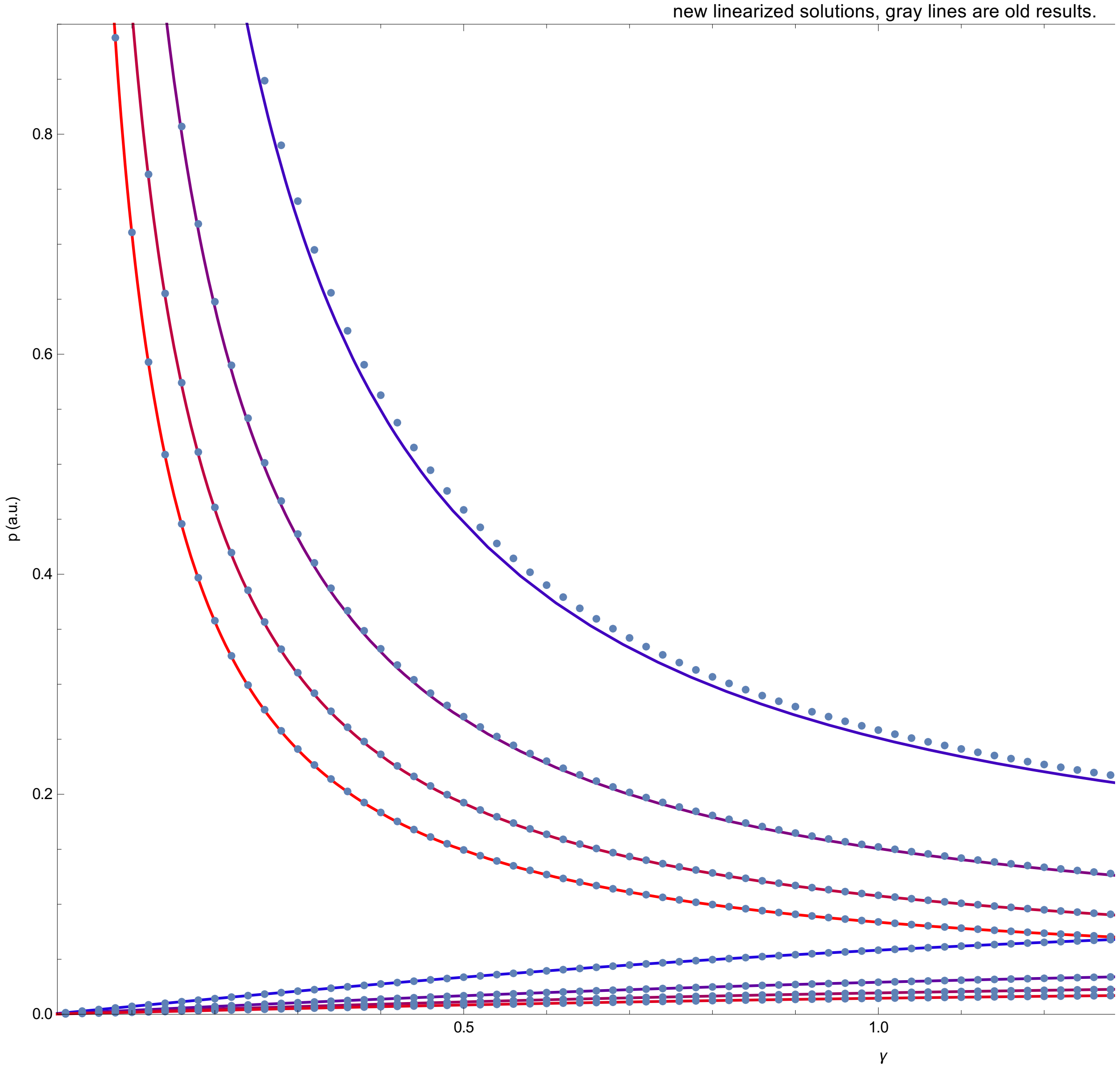
```
Block[{F = 0.05, κ = 1}, Show[Table[
    ParametricPlot[
     Tooltip[
      {ω κ/F, getFullReducedLinearizedTransition[n, F, ω, κ]}
      , n]
     , {ω, 0.0, 0.5}, PlotStyle → Blend[{Blue, Red}, n / 8]
     , Frame → True, PlotRangePadding → None, Axes → False,
     AspectRatio → 0.6, ImageSize → 800, FrameLabel → {"γ", "ωp/F"}
     , PlotLabel → "κ=1, F=0.05, ω∈[0,0.5]. Dots are exact solutions, coloured
        lines are\n new linearized solutions, gray lines are old results."
    ]
    , {n, 1, 8}] ~Join~ Table[
    ParametricPlot[
     Tooltip[
      {ω κ/F, getReducedLinearizedTransition[n, ω κ/F]}
      , n]
     , {ω, 0.0, 0.5}, PlotStyle → Gray
    ]
    , {n, 1, 8}] ~Join~ {
    ListPlot[
     Flatten[Table[
       {ω κ/F, ω pz/F} /. getClassicalTransition[Range[8], {F, ω, κ}]
       , {ω, 0.01, 0.5, 0.01}], 1]
     , PlotStyle → PointSize[Medium]
    ]
   }, PlotRange → {{0.01, 10}, {0, 0.9}}]]
```

κ=1, F=0.05, ω∈[0,0.5]. Dots are exact solutions, coloured lines are new linearized solutions, gray lines are old results.



```
Block[{F = 0.05, κ = 1}, Show[Table[
    ParametricPlot[
     Tooltip[
      {ω κ/F, F/ω getFullReducedLinearizedTransition[n, F, ω, κ]}
      , n]
     , {ω, 0.0, 0.1}, PlotStyle → Blend[{Blue, Red}, n / 8]
     , Frame → True, PlotRangePadding → None, Axes → False,
     AspectRatio → 0.6, ImageSize → 1000, FrameLabel → {"γ", "p (a.u.)"}
     , PlotLabel → "κ=1, F=0.05, ω∈[0,0.5]. Dots are exact solutions, coloured
        lines are\n new linearized solutions, gray lines are old results."
    ]
    , {n, 1, 8}]~Join~{
    ListPlot[
     Flatten[Table[
       {ω κ/F, F/ω ω pz/F} /. getClassicalTransition[Range[8], {F, ω, κ}]
       , {ω, 0.001, 0.1, 0.001}], 1]
     , PlotStyle → PointSize[Medium]
    ]
   }, PlotRange → {{0.01, 2}, {0, 0.9}}]]
```

κ=1, F=0.05, $\omega\in$[0,0.5]. Dots are exact solutions, coloured lin
new linearized solutions, gray lines are old results.



Trajectories

complexTrajectory

Returns $\boldsymbol{r}_{\text{cl}}(t) = \int_{t_s}^{t_r} \boldsymbol{p} + \boldsymbol{A}(\tau)\, d\tau$.

```
complexTrajectory::usage =
  "complexTrajectory[t,{px,py,pz},{F,ω,κ}] Returns the vector-valued
    complex trajectory r_cl(t)=∫_{t_s}^t (p+A(τ))dτ.

complexTrajectory[t,pz,{F,ω,κ}] Returns the z
    component of the complex trajectory z_cl(t)=∫_{t_s}^t (p_z+A(τ))dτ.";

rInit::usage =
  "rInit is an option for complexTrajectory and classicalTrajectory which
    specifies the initial position for the trajectory at time t_s.";
zInit::usage = "zInit is an option for complexTrajectory and classicalTrajectory
    which specifies the initial z position for the trajectory at time t_s.";
forcets::usage = "forcets is an option for complexTrajectory and
    classicalTrajectory which specifies a start time t_s to
    use for the trajectory, or uses the Automatic one.";
Protect[rInit, zInit, forcets];

Begin["`Private`"];
Options[complexTrajectory] = {zInit → 0, rInit → {0, 0, 0}, forcets → Automatic};
complexTrajectory[t_, pz_, {F_, ω_, κ_}, OptionsPattern[]] :=
 With[{tss = If[OptionValue[forcets] === Automatic,
     ts[{0, 0, pz}, {F, ω, κ}], OptionValue[forcets]]},
  OptionValue[zInit] + pz (t - tss) + F/ω^2 (Cos[ω t] - Cos[ω tss])
 ]
complexTrajectory[t_, {px_, py_, pz_}, {F_, ω_, κ_}, OptionsPattern[]] :=
 With[{tss = If[OptionValue[forcets] === Automatic,
     ts[{px, py, pz}, {F, ω, κ}], OptionValue[forcets]]},
  OptionValue[rInit] + {px, py, pz} (t - tss) + {0, 0, 1} F/ω^2 (Cos[ω t] - Cos[ω tss])
 ]
End[];
```

classicalTrajectory

Returns $\mathrm{Re}[\boldsymbol{r}_{\mathrm{cl}}(t)] = \mathrm{Re}\left[\int_{t_s}^{t_r} \boldsymbol{p} + \boldsymbol{A}(\tau)\, d\tau\right]$.

```
classicalTrajectory::usage =
  "classicalTrajectory[t, {px, py, pz}, {F, ω, κ}] Returns the real part of
    the vector-valued complex trajectory, Re(r_cl(t))=Re(∫_{t_s}^t (p+A(τ))dτ).

classicalTrajectory[t, pz, {F, ω, κ}] Returns the real part of the z
    component  of the complex trajectory, Re(z_cl(t))=Re(∫_{t_s}^t (p_z+A(τ))dτ).";
```

```
Begin["`Private`"];
classicalTrajectory[t_, pz_, {F_, ω_, κ_}, OptionsPattern[zInit → 0]] :=
 Re[complexTrajectory[t, pz, {F, ω, κ}, zInit → OptionValue[zInit]]]
classicalTrajectory[t_?NumericQ, {px_, py_, pz_},
  {F_, ω_, κ_}, OptionsPattern[rInit → 0]] :=
 Re[complexTrajectory[t, {px, py, pz}, {F, ω, κ}, rInit → OptionValue[rInit]]]
End[];
```

Closest Approach times

Classical t_{CA}s

classicalClosestApproach

Returns a list of the classical closest approach times in the specified Range, in atomic units. These are solutions of the equation $\mathrm{Re}[\boldsymbol{r}_{\mathrm{cl}}(t)]\cdot\boldsymbol{v}(t) = \mathrm{Re}\left[\boldsymbol{r}_{\mathrm{init}} + \int_{t_s}^{t}\boldsymbol{p}+\boldsymbol{A}(\tau)\,d\tau\right]\cdot\boldsymbol{v}(t) = 0$ and are all real valued.

```
classicalClosestApproach::usage =
  "classicalClosestApproach[{px, py, pz}, {F, ω, κ}] Returns tCA, such
    that Re[rcl(tCA)]·v(tCA)=0, for the given momentum and parameters.
    Accepts \"rules\" as an option, as well as \"Range\" in the
    format {t1, t2}, where both can contain the laser period \"T\".";
```

```
Begin["`Private`"];
Options[classicalClosestApproach] = {"rules" → Automatic, "Range" → {0, 2 "T"}};
classicalClosestApproach[{px_, py_, pz_}, {F_, ω_, κ_}, OptionsPattern[]] := Module[
  {tstart, zinit},
  tstart = If[NumberQ[OptionValue["rules"]],
    "t0" /. OptionValue["rules"],
    Re[1/ω ArcSin[ω/F (pz + ⅈ Sqrt[κ^2 + px^2 + py^2])]]
   ];
  zinit = F/ω^2 Cos[ω tstart] - F/ω^2 Re[Cos[ArcSin[ω/F (pz + ⅈ Sqrt[κ^2 + px^2 + py^2])]]];
  If[Length[#] > 0, t /. #, {}] &@Quiet[
    NSolve[{
      {px, py, pz - F/ω Sin[ω t]}.classicalTrajectory[t, {px, py, pz}, {F, ω, κ}] == 0,
      Evaluate[OptionValue["Range"][[1]] < t < OptionValue["Range"][[2]] /. {"T" → 2π/ω}]
     }, t]
   ]
 ]
End[];
```

rDotV

Returns the value of $\mathrm{Re}[\boldsymbol{r}_{\mathrm{cl}}(t)]\cdot\boldsymbol{v}(t) = \mathrm{Re}\left[\boldsymbol{r}_{\mathrm{init}} + \int_{t_s}^{t}\boldsymbol{p}+\boldsymbol{A}(\tau)\,d\tau\right]\cdot\boldsymbol{v}(t)$ for the specified time, momentum and parameters. Useful mainly as a cleaner way to plot its zero contours - i.e. the surfaces formed by the t_{CA} on different geometrical spaces.

```
rDotV::usage = "rDotV[t, px, pz, {F, ω, κ}] Returns the
    classical r(t)·v(t) for the given momentum and parameters.";
```

```
Begin["`Private`"];
rDotV[t_, {px_, py_, pz_}, {F_, ω_, κ_}] :=
 Module[{tss, zinit = F/ω^2 (1 - Sqrt[1 + (κ ω/F)^2])},
  tss = If[NumberQ[OptionValue["rules"]],
    "t0" /. OptionValue["rules"],
    Re[1/ω ArcSin[ω/F (pz + ⅈ Sqrt[κ^2 + px^2 + py^2])]]
   ];
  (px^2 + py^2) (t - tss) + (pz (t - tss) + F/ω^2 (Cos[ω t] - Cos[ω tss]) + zinit) (pz - F/ω Sin[ω t])
 ]
End[];
```

Two-dimensional version memoized for efficiency:

```
Begin["`Private`"];
rDotV[t_, px_, pz_, {F_, ω_, κ_}] :=
 rDotV[t, px, pz, {F, ω, κ}] = rDotV[t, {px, 0, pz}, {F, ω, κ}]
End[];
```

d2r2

Returns the second derivative $\frac{d^2}{dt^2}\left[\boldsymbol{r}_{\mathrm{cl}}(t)^2\right] = \frac{d^2}{dt^2}\left[\mathrm{Re}\left(\boldsymbol{r}_{\mathrm{init}} + \int_{t_s}^{t}\boldsymbol{p}+\boldsymbol{A}(\tau)\,d\tau\right)^2\right]$.

```
d2r2::usage =
  "d2r2[t, {px, py, pz}, {F, ω, κ}] Returns the classical second time derivative
    d²/dt² r_cl² at the given momentum and parameers.";
```

```
Begin["`Private`"];
d2r2[t_, {px_, py_, pz_}, {F_, ω_, κ_}] :=
 d2r2[t, {px, py, pz}, {F, ω, κ}] = 2 (Norm[{px, py, pz} - {0, 0, 1} F/ω Sin[ω t]]^2 -
     classicalTrajectory[t, pz, {F, ω, κ}] F Cos[ω t])
End[];
```

Quantum t_{CA}s

are the complex solutions of $\boldsymbol{r}_{cl}(t)\cdot\boldsymbol{v}(t) = \left(\boldsymbol{r}_{init} + \int_{t_s}^{t}\boldsymbol{p}+\boldsymbol{A}(\tau)\,d\tau\right)\cdot\boldsymbol{v}(t) = 0$.

allQuantumClosestApproachTimes

```
allQuantumClosestApproachTimes::usage =
  "allQuantumClosestApproachTimes[{px, py, pz}, {F, ω, κ}, {xinit, yinit, zinit}]
    returns the quantum tCAs as a list of complex values. It accepts as
    options an explicit \"ts\" and a \"Range\", set to {-2iτ, T+2iτ} by
    default, as well as all the options of EPToolbox`FindComplexRoots.";
tCA::usage = "tCA represents a closest approach time tCA.";

Begin["`Private`"];
Options[allQuantumClosestApproachTimes] =
  Join[Options[FindComplexRoots], {"rules" → Automatic, "Range" → Automatic}];
allQuantumClosestApproachTimes[{po_, py_, pp_}, {F_, ω_, κ_},
  {xinit_, yinit_, zinit_}, options : OptionsPattern[]] := Module[
  {tss, range, rules},
  tss = If[OptionValue["rules"] === Automatic,
    ts[pp, κ, ω, F, po, py], "ts" /. OptionValue["rules"]];
  rules = If[OptionValue["rules"] === Automatic,
    {"tκ" → tss - i/κ^2, "ts" → tss, "t0" → Re[tss], "τ" → Im[tss], "T" → 2 π / ω},
    OptionValue[rules]
   ];
  range = Which[
    MatchQ[OptionValue["Range"] /. rules, {a_?NumericQ, b_?NumericQ} /; Im[b - a] ≤ 0],
    (OptionValue[Range] /. rules) + {-2 i Im[tss], 2 i Im[tss]},
    MatchQ[OptionValue["Range"] /. rules, {_?NumericQ, _?NumericQ}],
    (OptionValue[Range] /. rules),
    True, {-2 i Im[tss], 2 π/ω + 2 i Im[tss]}
   ];
  Sort@FindComplexRoots[
    2 ({xinit, yinit, zinit} + {po, py, pp} (tCA - tss) +
        {0, 0, F/ω^2 (Cos[ω tCA] - Cos[ω tss])}).{po, py, pp - F/ω Sin[ω tCA]} == 0
    , {tCA, range[[1]], range[[2]]}
    , Sequence @@ FilterRules[{options}, Options[FindComplexRoots]]
    , Seeds → 200
    , Tolerance → 10^(4 - $MachinePrecision)
   ]
 ]
End[];
```

makeCircuitTCAsFromCircuit

Takes a ready-made circuit, in the format $\{\{n_1, \{p_{x\,1}, p_{y\,1}, p_{z\,1}\}\}, \ldots, \{n_N, \{p_{x\,N}, p_{y\,N}, p_{z\,N}\}\}\}$, and calculates all the relevant t_{CA}s for it, returning the tags and the momentum in the output, which is of the form $\{\{n_1, \{p_{x\,1}, p_{y\,1}, p_{z\,1}\}, t_{\text{CA}\,1,1}\}, \{\{n_1, \{p_{x\,1}, p_{y\,1}, p_{z\,1}\}, t_{\text{CA}\,1,2}\}, \ldots, \{\{n_N, \{p_{x\,N}, p_{y\,N}, p_{z\,N}\}, t_{\text{CA}\,N,k}\}\}$.

```
makeTCAsFromCircuit::usage =
  "makeTCAsFromCircuit[{{n1, {px1, py1, pz1}}, ..., {nN, {pxnN, pynN,
    pznN}}}, {F, ω, κ}, {xinit, yinit, zinit}] Calculates the tCAs for
    the given circuit and parameters. The ni can be any tags which are
    returned with the output, which is of the form {{n1, {px1, py1, pz1},
    tCA11}, {n1, {px1, py1, pz1}, tCA12}, ..., {nN, {pxnN, pynN, pznN},
    tCAnNk}}, with all the appropriate tCA in separate entries. Same
    \"rules\" and \"Range\" options as allQuantumClosestApproachTimes.";
```

```
Begin["`Private`"]
Options[makeTCAsFromCircuit] =
  Join[{"rules" → Automatic, OptionValue["Range"] → Automatic,
    PlotRange → Automatic}, Options[allQuantumClosestApproachTimes]];
makeTCAsFromCircuit[circuit_, {F_, ω_, κ_}, {xinit_, yinit_, zinit_},
  options : OptionsPattern[]] := Module[
  {range, rules, tss, n, pvec},
  Flatten[ParallelTable[
    {n, pvec} = element;
    Needs["EPToolbox`",
     "/home/episanty/Work/CQD/Project/Code/EPToolbox/EPToolbox/EPToolbox.m"];
    tss = If[OptionValue["rules"] === Automatic,
      ts[pvec[[1]], κ, ω, F, pvec[[1]]], "ts" /. OptionValue["rules"]];
    rules = If[OptionValue["rules"] === Automatic,
      {"tκ" → tss - i / κ^2, "ts" → tss, "t0" → Re[tss], "τ" → Im[tss], "T" → 2 π / ω},
      OptionValue[rules]
     ];
    range = Automatic;
    range = Which[
      MatchQ[OptionValue["Range"] /. rules, {_?NumericQ, _?NumericQ}],
      OptionValue["Range"] /. rules,
      MatchQ[OptionValue[PlotRange] /. rules, {{_?NumericQ, _?NumericQ},
         {_?NumericQ, _?NumericQ}}], Complex @@@ (OptionValue[PlotRange]^T /. rules),
      True, {-2 i Im[tss], 2 π/ω + 2 i Im[tss]}
     ];
    (*ugly logic inside the Table because
     range depends on tss which depends on p*)
    {n, pvec, tCA} /. allQuantumClosestApproachTimes[
      {pvec[[1]], 0, pvec[[2]]}, {F, ω, κ}, {xinit, yinit, zinit}
      , Sequence @@ FilterRules[{options}, Options[allQuantumClosestApproachTimes]],
      "Range" → range
     ]
    , {element, circuit}], 1]
 ]
End[]
```

```
ARMSupport`Private`

ARMSupport`Private`
```

makeTCAsFromRange

Takes a specific range of momentum and gets all the relevant t_{CA}s for a rectangular grid of those specifications.

```
makeTCAsFromRange::usage =
  "makeTCAsFromRange[{pomin, pomax, δpo}, {ppmin, ppmax, δpp}, fixedMomenta,
    {F, ω, κ}, {xinit, yinit, zinit}, \"Range\"→{t1, t2}] Returns
    a list with elements of the form {{po, py, pp}, tCA} for a
    rectangular grid in momentum with the given spans and separations.
    fixedMomenta should be a list of replacement rules such as {py→0}.";

Begin["`Private`"];
Options[makeTCAsFromRange] = Options[allQuantumClosestApproachTimes];
makeTCAsFromRange[{pomin_, pomax_, δpo_}, {ppmin_, ppmax_, δpp_}, fixedMomenta_,
  {F_, ω_, κ_}, {xinit_, yinit_, zinit_}, options : OptionsPattern[]] := Module[
  {range, rules, tss},
  tss = If[OptionValue["rules"] === Automatic,
    ts[pp, κ, ω, F, po, Global`py], "ts" /. OptionValue["rules"]];
  rules = If[OptionValue["rules"] === Automatic,
    {"tκ" → tss - i / κ^2, "ts" → tss, "t0" → Re[tss], "τ" → Im[tss], "T" → 2 π / ω},
    OptionValue[rules]
   ];
  Flatten[
   Table[
    range = Which[
       MatchQ[OptionValue["Range"] /. rules /. fixedMomenta,
        {_?NumericQ, _?NumericQ}], OptionValue["Range"] /. rules,
       MatchQ[OptionValue["Range"] /. rules /. fixedMomenta,
        {{_?NumericQ, _?NumericQ}, {_?NumericQ, _?NumericQ}}],
       Complex @@@ (OptionValue["Range"]^T /. rules),
       True, {-2 i Im[tss], 2 π/ω + 2 i Im[tss]}
      ] /. fixedMomenta;
    (*ugly logic inside the Table
     because range depends on tss which depends on p*)
    {{po, Global`py, pp} /. fixedMomenta, tCA} /. allQuantumClosestApproachTimes[
      {po, Global`py, pp} /. fixedMomenta
      , {F, ω, κ}, {xinit, yinit, zinit}
      , "Range" → range,
      Sequence @@ FilterRules[{options}, Options[allQuantumClosestApproachTimes]]
     ]
    , {po, pomin, pomax, δpo}, {pp, ppmin, ppmax, δpp}]
   , {1, 2, 3}]
 ]
End[];
```

Note the messy Global`py. This function should really be restructured to avoid the context conflicts this sort of mostly hackishly fixes.

closestApproachTimesPath

uses magic to choose the appropriate t_{CA}'s the integration contour should pass through. Output is the same as allQuantumClosestApproachTimes.

```
closestApproachTimesPath::usage =
  "closestApproachTimesPath[{px, py, pz}, {F, ω, κ}, {xinit, yinit,
    zinit}] Returns a selected and ordered list of complex tCAs as
    replacement rules, in atomic units. Accepts the same \"rules\"
    and \"Range\" options as allQuantumClosestApproachTimes.";
v2Tolerance::usage = "v2Tolerance is an option for closestApproachTimesPath
   which determines the tolerance v2tol to be used when selecting tCAs
   for the path. Time tCA is included in the path if Re[v[tCA]^2]≥-v2tol.";
Protect[v2Tolerance];

Begin["`Private`"]
Options[closestApproachTimesPath] = Join[{v2Tolerance → Automatic},
   Options[allQuantumClosestApproachTimes], Options[ListPlot]];
closestApproachTimesPath[{po_, py_, pp_}, {F_, ω_, κ_},
  {xinit_, yinit_, zinit_}, options : OptionsPattern[]] := Module[
  {tss, r, v, range, rules, v2tol},
  tss = If[OptionValue["rules"] === Automatic,
    ts[pp, κ, ω, F, po, py], "ts" /. OptionValue["rules"]];
  rules = If[OptionValue["rules"] === Automatic,
    {"tκ" → tss - i / κ^2, "ts" → tss, "t0" → Re[tss], "τ" → Im[tss], "T" → 2 π / ω},
    OptionValue[rules]
   ];
  v2tol = Which[OptionValue[v2Tolerance] === Automatic,
    10^-8, True, OptionValue[v2Tolerance]];
  r[tt_] := ({xinit, yinit, zinit} + {po, py, pp} (tt - tss) +
      {0, 0, F/ω^2 (Cos[ω tt] - Cos[ω tss])});
  v[tt_] := ({po, py, pp} + {0, 0, - F/ω Sin[ω tt]});
  range = Which[
    MatchQ[OptionValue["Range"] /. rules, {_?NumericQ, _?NumericQ}],
    OptionValue["Range"] /. rules,
    MatchQ[OptionValue[PlotRange] /. rules, {{_?NumericQ, _?NumericQ},
       {_?NumericQ, _?NumericQ}}], Complex @@@ (OptionValue[PlotRange]^T /. rules),
    True, {-2 i Im[tss], 2 π/ω + 2 i Im[tss]}
   ];
  Select[
   Sort[
    allQuantumClosestApproachTimes[{po, py, pp}, {F, ω, κ}, {xinit, yinit, zinit}
     , "Range" → range
```

```
      , Sequence @@ FilterRules[{options}, Options[allQuantumClosestApproachTimes]]
      , Tolerance → 10 ^ (4 - $MachinePrecision)],
     ((Re[tCA] /. #1) < (Re[tCA] /. #2)) &
    ],
    (Or[
        And[1/4 2π/ω < Re[tCA] < 3/4 2π/ω, Im[tCA] > 0],
        And[-1/4 2π/ω < Re[tCA] < 1/4 2π/ω, Im[tCA] ≥ 0, Im[Sin[ω tCA]] < ω κ / F],
        And[(-0.3 Im[tss] ≤ Im[tCA] < Im[tss] - 1/κ^2),
         (Re[tCA] - 0.1 2π/ω > Re[tss]), (Re[v[tCA].v[tCA]] ≥ -v2tol)]
       ] /. # &)
   ]
  ]
End[]
(*closestApproachTimesPath[{0.05,0,1.2},
 stdpars,{0,0,0},"Range"→{-5ⅈ,5.6"T"+30ⅈ}]*)
```

```
ARMSupport`Private`

ARMSupport`Private`
```

Amplitude-related functions

Volkov exponent

```
volkovExponent::usage =
  "volkovExponent[{po, py, pp}, {F, ω, κ}] calculates Re(ⅈ∫_0^ts (I_p+1/2(p+A(τ))^2)ⅆτ).";
```

```
Begin["`Private`"]
(*FullSimplify[Re[
  ⅈ Integrate[κ^2/2+1/2 (po^2+py^2)+1/2 (pp-F/ω Sin[ω t])^2,{t,0,1/ω ArcSin[ω/F (pp+ⅈ Sqrt[κ^2+po^2+py^2])]}]/.
   {Sin[2u_]→2Sin[u]Cos[u]}
 ]]*)
volkovExponent[{po_, py_, pp_}, {F_, ω_}, κ_] :=
```

$$-\frac{1}{8}\,\mathrm{Im}\left[\frac{1}{\omega^3}\left(2\,F\,\omega\left(-4\,pp+3\,pp\sqrt{1+\frac{\left(-i\,pp+\sqrt{po^2+py^2+\kappa^2}\right)^2\omega^2}{F^2}}-i\sqrt{po^2+py^2+\kappa^2}\sqrt{1+\frac{\left(-i\,pp+\sqrt{po^2+py^2+\kappa^2}\right)^2\omega^2}{F^2}}\right)+2\left(F^2+2\left(po^2+pp^2+py^2+\kappa^2\right)\omega^2\right)\mathrm{ArcSin}\left[\frac{\left(pp+i\sqrt{po^2+py^2+\kappa^2}\right)\omega}{F}\right]\right)\right]$$

```
volkovExponent[{po_, py_, pp_}, {F_, ω_, κ_}] := volkovExponent[{po, py, pp}, {F, ω}, κ]
End[]
ARMSupport`Private`

ARMSupport`Private`
```

coulombCorrection

Numerically integrates $\int_C \frac{1}{\sqrt{r_{\mathrm{cl}}(t)^2}}\,dt$ over the specified complex integration path C. Sows parameters when integration errors are encountered, and allows for softening of the Coulomb kernel if required.

Definition

```
coulombCorrection::usage =
  "coulombCorrection[{px, py, pz}, {F, ω, κ}, path] Calculates the Coulomb
    correction integral over the specified path. The path is a list
    which may contain \"tκ\", \"ts\", \"t0\", \"τ\", \"tCApath\"
    and \"T\", which will be replaced by the appropriate points.";
coulombCorrection::intErrors = "Integration errors obtained
    at input {{po, py, pp}, {F, ω, κ}, path}=`1`";
Softening::usage = "Softening is an option for coulombCorrection which
    specifies whether the Coulomb kernel should be softened by
    a length σ. It is set by default to None (σ=0), and it can
    be changed to Automatic (σ=1/κ) or a numeric value for σ.";
ReportingFunction::usage = "ReportingFunction is an option for coulombCorrection
    to specify the reporting of error-producing inputs. It should
    speficy a function f, set by default to Sow, which will be called as
    f[{{po, py, pp}, {F, ω, κ}, path}] if the inputs produce any errors
    during the NIntegrate call. To print to a file use ReportToFile.";
ReportToFile::usage = "ReportToFile[directory, file] returns a
    function which can be used as a value for ReportingFunction
    inside coulombCorrection.\n\nReportToFile[directory,
    file][expr] adds a line with expr (properly parsed to ASCII
    for spaces, backslashes and quote marks) to directory/file.";
```

```
Begin["`Private`"]
If[$KernelID == 0, SetSharedFunction[Sow]];
Quiet[ReportingFunction = ReportingFunction; Softening = Softening;]
Protect[ReportingFunction]; Protect[Softening];

Options[coulombCorrection] = Join[
    {Softening → None, ReportingFunction → Sow}, Options[closestApproachTimesPath]];
coulombCorrection[{po_, py_, pp_}, {F_, ω_, κ_}, path_: {"tκ", "t0"},
   options : OptionsPattern[]] := Block[
   {tss, iterator, rules, range, tCApath, int, σ},
   σ = Which[NumberQ[OptionValue[Softening]],
     OptionValue[Softening], OptionValue[Softening] === Automatic, 1 / κ, True, 0];
   (*Coulomb softening*)
   tss = ts[pp, κ, ω, F, po, py];
   rules = {"tκ" → tss - ⅈ/κ^2, "ts" → tss, "t0" → Re[tss], "τ" → Im[tss], "T" → 2 π / ω};
   range = ({Re[First[path]] - 2 ⅈ "τ", Re[Last[path]] + 2 ⅈ "τ"} /. rules);
   If[
    ! FreeQ[path, "tCApath"],
    tCApath = Chop[tCA /. closestApproachTimesPath[{po, py, pp}, {F, ω, κ}, {0, 0, 0},
        Sequence @@ FilterRules[{options}, Options[closestApproachTimesPath]],
        "Range" → range]];
    If[Length[tCApath] > 0,
     AppendTo[rules, "tCApath" → Apply[Sequence, tCApath]],
     AppendTo[rules, "tCApath" → (## &[])]
    ]
   ]; (*Print[rules];*)
   iterator = {t, Sequence @@ Evaluate[path /. rules]};
   (*Print[iterator];*)
   Check[
    int = NIntegrate[
      -((po^2 + py^2) (t - tss)^2 + (pp (t - tss) + F/ω^2 (Cos[ω t] - Cos[ω tss]))^2 + σ^2)^(-1/2),
      Evaluate@iterator],
    OptionValue[ReportingFunction][Chop[{{po, py, pp}, {F, ω, κ}, path}]];
    Message[coulombCorrection::intErrors, Chop[{{po, py, pp}, {F, ω, κ}, path}]]; int
   ]
  ]
End[]
ARMSupport`Private`
```

```
Image::imgarray : The specified argument
    {{{255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255,
        255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 254}, {252, 252
        , 252}, {254, 254, 255}, {255, 255, 255}, {255, 255, 255}, ≪20≫, {255, 255, 255}, {255, 255, 255}, {255,
        255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255
        , 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, {255, 255, 255}, ≪100≫}, ≪
      49≫, ≪50≫} should be an array of rank 2 or 3 with machine–sized numbers. ≫

ARMSupport`Private`
```

```
Begin["`Private`"]
ReportToFile[directory_, file_] :=
 Function[expr, Run["cd " <> directory <> " && echo " <>
    StringReplace[ToString[expr /. {s_String :> StringJoin["\"", s, "\""]},
      CharacterEncoding → "ASCII"], {" " → "\\ ", "\\" → "\\\\", "\"" → "\\\""}] <>
    " >> " <> StringReplace[file, {" " → "\\ ", "\\" → "\\\\", "\"" → "\\\""}]]]
End[]
```

```
ARMSupport`Private`

ARMSupport`Private`
```

Tests of the error handling

For a single evaluation

```
Reap[
 coulombCorrection[{2, 0, 0}, {0.05, 0.055, 1.007}, {"tκ", "t0"}]
]
```

```
NIntegrate::slwcon :
  Numerical integration converging too slowly; suspect one of the following: singularity, value of
      the integration is 0, highly oscillatory integrand, or WorkingPrecision too small. ≫

NIntegrate::ncvb :
  NIntegrate failed to converge to prescribed accuracy after 9 recursive bisections in t near {t} = {0. + 25.8533 ⅈ}.
      NIntegrate obtained 2.67192 + 2.68596 ⅈ and 0.061081270393484745`
      for the integral and error estimates. ≫

coulombCorrection::intErrors :
  Integration errors obtained at input {{po, py, pp}, {F, ω, κ}, path}={{2, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}}

{2.67192 + 2.68596 ⅈ, {{{{2, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}}}}}
```

Test of the error handling inside a Parallelized Table.

```
(*This needs to be run before any parallel evaluation.*)
ParallelEvaluate[Needs["EPToolbox`",
   "/home/episanty/Work/CQD/Project/Code/EPToolbox/EPToolbox/EPToolbox.m"]];
```

```
Reap[
  ParallelTable[
   Quiet@coulombCorrection[{po, 0, 0}, {0.05, 0.055, 1.007}, {"tκ", "t0"}]
   , {po, -2, 2, 0.1}]
 ][[2, 1]]
(*Generates about 2 pages of errors if not
 Quieted. This shows the Reaped trouble inputs only.*)
```

```
{{{-1.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.4, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.1, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-0.8, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-2., 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.6, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.3, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1., 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-0.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.9, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.5, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.2, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-0.9, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-0.6, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{0.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{0.9, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.8, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{0.6, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{0.8, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1., 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.9, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.1, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.3, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.5, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{2., 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.2, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.4, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.6, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.8, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}}}
```

Print errors to file:

```
coulombCorrection[{0.01, 0, 0.5}, stdpars, {"tκ", 2 "T"},
 ReportingFunction → ReportToFile[NotebookDirectory[], "test.txt"]]
```

```
NIntegrate::slwcon :
  Numerical integration converging too slowly; suspect one of the following: singularity, value of
     the integration is 0, highly oscillatory integrand, or WorkingPrecision too small. ≫

NIntegrate::ncvb : NIntegrate failed to converge to prescribed accuracy
     after 9 recursive bisections in t near {t} = {75.9294 + 11.9 ⅈ}. NIntegrate obtained
     -7.48256 + 3.89417 ⅈ and 0.004771979130395705` for the integral and error estimates. ≫

coulombCorrection::intErrors :
  Integration errors obtained at input {{po, py, pp}, {F, ω, κ}, path}={{0.01, 0, 0.5}, {0.05, 0.055, 1.007}, {tκ, 2 T}}

-7.48256 + 3.89417 ⅈ
```

and then reimport from the file

```
ToExpression[Import[NotebookDirectory[] <> "test.txt"]]
coulombCorrection @@ %
```

```
{{0.01, 0, 0.5}, {0.05, 0.055, 1.007}, {tκ, 2 T}}

NIntegrate::slwcon :
  Numerical integration converging too slowly; suspect one of the following: singularity, value of
     the integration is 0, highly oscillatory integrand, or WorkingPrecision too small. ≫

NIntegrate::ncvb : NIntegrate failed to converge to prescribed accuracy
     after 9 recursive bisections in t near {t} = {75.9294 + 11.9 ⅈ}. NIntegrate obtained
     -7.48256 + 3.89417 ⅈ and 0.004771979130395705` for the integral and error estimates. ≫

coulombCorrection::intErrors :
  Integration errors obtained at input {{po, py, pp}, {F, ω, κ}, path}={{0.01, 0, 0.5}, {0.05, 0.055, 1.007}, {tκ, 2 T}}

-7.48256 + 3.89417 ⅈ
```

Printing to file from inside a parallelized environment. Note that parallelized kernels have no FrontEnd and therefore cannot access NotebookDirectory[].

```
Block[{directory = NotebookDirectory[]},
 ParallelTable[
   Quiet@coulombCorrection[{po, 0, 0}, {0.05, 0.055, 1.007},
     {"tκ", "t0"}, ReportingFunction → ReportToFile[directory, "test.txt"]]
   , {po, -2, 2, 0.1}];
]
```

```
ToExpression /@ Import[NotebookDirectory[] <> "test.txt", "List"]
coulombCorrection @@@ %
```

```
{{{-1.4, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.1, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-0.8, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.3, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.6, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-2., 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1., 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.2, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-0.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.5, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-0.9, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.9, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-0.6, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{0.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{0.9, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.1, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{-1.8, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.5, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{0.8, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1., 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.3, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.2, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.8, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{0.6, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.6, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.4, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{1.9, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}},
 {{2., 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}}}
```

```
NIntegrate::ncvb :
  NIntegrate failed to converge to prescribed accuracy after 9 recursive bisections in t near {t} = {0. + 18.639 ⅈ}.
      NIntegrate obtained 2.37932 + 3.44132 ⅈ and 0.07476695623809278`
      for the integral and error estimates. ≫

coulombCorrection::intErrors :
  Integration errors obtained at input {{po, py, pp}, {F, ω, κ}, path}={{–1.4, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}}

NIntegrate::ncvb :
  NIntegrate failed to converge to prescribed accuracy after 9 recursive bisections in t near {t} = {0. + 22.5694 ⅈ}.
      NIntegrate obtained 2.5598 + 3.02543 ⅈ and 0.05200239804482814`
      for the integral and error estimates. ≫

coulombCorrection::intErrors :
  Integration errors obtained at input {{po, py, pp}, {F, ω, κ}, path}={{–1.7, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}}

NIntegrate::slwcon :
  Numerical integration converging too slowly; suspect one of the following: singularity, value of
      the integration is 0, highly oscillatory integrand, or WorkingPrecision too small. ≫
```

```
NIntegrate::ncvb :
  NIntegrate failed to converge to prescribed accuracy after 9 recursive bisections in t near {t} = {0. + 13.7753 i}.
      NIntegrate obtained 2.12149 + 3.84551 i and 0.040628351055489696`
      for the integral and error estimates. ≫

General::stop : Further output of NIntegrate::ncvb will be suppressed during this calculation. ≫

coulombCorrection::intErrors :
  Integration errors obtained at input {{po, py, pp}, {F, ω, κ}, path}={{-1.1, 0, 0}, {0.05, 0.055, 1.007}, {tκ, t0}}

General::stop : Further output of coulombCorrection::intErrors will be suppressed during this calculation. ≫

NIntegrate::slwcon :
  Numerical integration converging too slowly; suspect one of the following: singularity, value of
      the integration is 0, highly oscillatory integrand, or WorkingPrecision too small. ≫

NIntegrate::slwcon :
  Numerical integration converging too slowly; suspect one of the following: singularity, value of
      the integration is 0, highly oscillatory integrand, or WorkingPrecision too small. ≫

General::stop : Further output of NIntegrate::slwcon will be suppressed during this calculation. ≫

{2.37932 + 3.44132 i, 2.5598 + 3.02543 i, 2.12149 + 3.84551 i,
 1.60378 + 4.39982 i, 2.33519 + 3.5208 i, 2.50409 + 3.16691 i, 2.67192 + 2.68596 i,
 1.98033 + 4.00272 i, 2.23833 + 3.67126 i, 1.34139 + 4.5624 i, 2.47792 + 3.24468 i,
 1.85702 + 4.16404 i, 2.63455 + 2.79829 i, 0.931087 + 4.7784 i,
 1.34139 + 4.5624 i, 1.85702 + 4.16404 i, 2.12149 + 3.84551 i, 2.5598 + 3.02543 i,
 2.61225 + 2.91755 i, 2.47792 + 3.24468 i, 1.60378 + 4.39982 i, 1.98033 + 4.00272 i,
 2.33519 + 3.5208 i, 2.23833 + 3.67126 i, 2.61225 + 2.91755 i, 0.931087 + 4.7784 i,
 2.50409 + 3.16691 i, 2.37932 + 3.44132 i, 2.63455 + 2.79829 i, 2.67192 + 2.68596 i}
```

SFT calculations

Preliminaries

This assigns the value 1 to the formally undefined expression 0^0, which in this context is always a special case of r^n for real r and integer n. However, it is important to note that this is (in principle) dangerous, and it can potentially cause errors in other notebooks that are running on the same kernel.

```
Unprotect[Power];
Power[0, 0] = 1;
Power[0. + 0. i, 0] = 1;
Power[0., 0] = 1;
Protect[Power];
```

Some functions for support. The velocity vector **vel** at t_s as a function of alignment angle and transverse momentum and the corresponding $\boldsymbol{q}$ vector.

```
vel::usage =
  "vel[θ,pₒ,py,κ] calculates the velocity vector p+A(ts) at the saddle point
    time ts, for the transverse momentum (pₒ,py), the target Ip=κ²/2, and
    an angle θ between the molecular axis and the laser polarization.";
qvec::usage = "qvec[a,θ,pₒ,py,κ] returns the normalized velocity
    q=a(p+A(ts)), with the inner part as for the
    function vel, for a the ARM boundary radius.";
Begin["`Private`"];
vel[θ_, po_, py_, κ_] :=
  (po {Cos[θ], 0, -Sin[θ]} + py {0, 1, 0} - ⅈ Sqrt[κ^2 + po^2 + py^2] {Sin[θ], 0, Cos[θ]});
qvec[a_, θ_, po_, py_, κ_] := a vel[θ, po, py, κ];
End[];
```

SFTnumeric

Uses parallelized memoization as per mm.se/q/1259

```
SFTnumeric::usage = "SFTnumeric[]";
SFTnumericParallelized::usage = "SFTnumericParallelized";
Begin["`Private`"];
SFTnumeric[qx_, qy_, qz_, b_, c_, nx_, ny_, nz_] :=
  With[{result = SFTnumericParallelized[qx, qy, qz, b, c, nx, ny, nz]},
   (SFTnumeric[qx, qy, qz, b, c, nx, ny, nz] = result) /; result =!= Null];
SFTnumeric[qx_, qy_, qz_, b_, c_, nx_, ny_, nz_] :=
 SFTnumericParallelized[qx, qy, qz, b, c, nx, ny, nz] =
  SFTnumeric[qx, qy, qz, b, c, nx, ny, nz] = NIntegrate[
    1/(2 π) ⅇ^(-ⅈ (qx Sin[θ] Cos[ϕ] + qy Sin[θ] Sin[ϕ] + qz Cos[θ])) Cosh[b Cos[θ]]
    (1 + c Cos[θ]^2) Cos[θ]^nz Sin[θ]^(nx+ny+1) Cos[ϕ]^nx Sin[ϕ]^ny
    , {θ, 0, π}, {ϕ, 0, 2 π}
    , Method → "MultidimensionalRule"
   ]
SetSharedFunction[SFTnumericParallelized];
SFTnumeric[{qx_, qy_, qz_}, b_, c_, nx_, ny_, nz_] :=
 SFTnumeric[qx, qy, qz, b, c, nx, ny, nz]
End[];
```

SFTanalytic

```
SFTanalytic::usage = "SFTanalytic[] ";
Begin["`Private`"];
SFTanalytic[qx_, qy_, qz_, b_, c_, nx_, ny_, nz_] := With[
  {ss = Function[s, Sqrt[qx^2 + qy^2 + (qz + s ⅈ b)^2]], j = SphericalBesselJ, n = nx + ny + nz},
  Sum[(-ⅈ)^(nx+ny+nz) qx^nx qy^ny (qz + s ⅈ b)^nz (((1 + c (nz + 1/2)/(n + 3/2)) j[n, ss[s]]/ss[s]^n -
      c ((qz + s ⅈ b)^2/(qx^2 + qy^2 + (qz + s ⅈ b)^2) - (nz + 1/2)/(n + 3/2)) j[n + 2, ss[s]]/ss[s]^n), {s, {1, -1}}]
 ]
SFTanalytic[{qx_, qy_, qz_}, b_, c_, nx_, ny_, nz_] :=
 SFTanalytic[qx, qy, qz, b, c, nx, ny, nz]
End[];
```

SFTasymptotic

AsymptoticBesselI

```
AsymptoticBesselI::usage = "AsymptoticBesselI";
Begin["`Private`"];
AsymptoticBesselI[n_, σ_, order_: 1] := Block[{n1, σ1},
  AsymptoticBesselI[n1_, σ1_, order] =
   Normal[Delete[Series[BesselI[n1, σ1], {σ1, ∞, order}], {2, 2}]];
  AsymptoticBesselI[n, σ, order]
 ]
End[];
```

This provides an appropriate asymptotic series for the spherical Bessel functions of the exact analytic SFT, which are in the modified-Bessel-function regime of the form $j_n(ⅈ\,\sigma)$.

It is important to note that the precise phrasing of this code is very important and it is overall very finick-y.This is because the naive command for the asymptotic series of the Bessel function gets the polynomial part correctly, but it returns subexponential terms which are not desired and in general not particularly correct:

```
Series[BesselI[n, σ], {σ, ∞, 1}]
```

$$e^{-\sigma}\left(e^{2\sigma}\left(\frac{\sqrt{\frac{1}{\sigma}}}{\sqrt{2\pi}}+O\left[\frac{1}{\sigma}\right]^{3/2}\right)+\left(\frac{ⅈ\, e^{ⅈ n\pi}\sqrt{\frac{1}{\sigma}}}{\sqrt{2\pi}}+O\left[\frac{1}{\sigma}\right]^{3/2}\right)\right)$$

Leaving aside the weird factorization, the $e^{-\sigma}e^{2\sigma}$ factor is correct but the pure exponential-decay factor in $e^{-\sigma}\times\text{poly}(\sigma)$ is not what we want for large σ. In some ways this is understandable as the half-integer modified Bessel functions come out in terms of hyperbolic sines and cosines,

```
BesselI[1 / 2, σ]
BesselI[11 / 2, σ]
```

$$\frac{\sqrt{\frac{2}{\pi}}\ \mathrm{Sinh}[\sigma]}{\sqrt{\sigma}}$$

$$\frac{\left(2+\frac{1890}{\sigma^4}+\frac{210}{\sigma^2}\right)\mathrm{Cosh}[\sigma]+\left(-\frac{1890}{\sigma^5}-\frac{840}{\sigma^3}-\frac{30}{\sigma}\right)\mathrm{Sinh}[\sigma]}{\sqrt{2\,\pi}\ \sqrt{\sigma}}$$

but in the asymptotic regime these go away, and they are explicitly ignored in e.g. DLMF 10.40.1. Moreover, it is apparently impossible to get Mathematica to produce the asymptotic series without those terms, even by providing suitable **Assumptions**. To deal with this the code uses a **Delete** statement on the offending terms, but that relies on the to-be-deleted terms being in part ⟦2,2⟧ of the outputof **Series**, which is liable to break if the output is reordered through whatever reason.

So: the above code as stated works, just be very careful with these things when modifying it.

SFTasymptotic

```
SFTasymptotic::usage = "SFTasymptotic[]";
Begin["`Private`"];
ClearAll[SFTasymptotic];
SFTasymptotic[poo_, pyy_, θθ_, bb_, cc_, κκ_, aa_, nx_, ny_, nz_, order_] :=
 Block[{po, py, θ, b, c, κ, a},
  SFTasymptotic[po_, py_, θ_, b_, c_, κ_, a_, nx, ny, nz, order] = Block[
    {n = nx + ny + nz, qx, qy, qz, σ, s, κa},
    σ = κa Sqrt[(1 + b^2/κa^2 - 2 s b/κa Sqrt[1 + po^2/κ^2 + py^2/κ^2] Cos[θ] + 2 s ⅈ b/κa po/κ Sin[θ])];

    {qx, qy, qz} = {κa (po Cos[θ] - ⅈ Sqrt[po^2 + py^2 + κ^2] Sin[θ])/κ,
      κa py/κ, κa (-ⅈ Sqrt[po^2 + py^2 + κ^2] Cos[θ] - po Sin[θ])/κ};
    ⅇ^(κ a) ExpToTrig[Sum[
       Normal[Series[
          (-ⅈ)^n ⅇ^(-κa) qx^nx qy^ny (qz + s ⅈ b)^nz Sqrt[π/2] ((1 + c (nz + 1/2)/(n + 3/2)) 1/σ^(n+1/2)
             AsymptoticBesselI[n + 1/2, σ, order + 1] - c ((qz + s ⅈ b)^2 + (nz + 1/2)/(n + 3/2) σ^2)
             1/σ^(n+5/2) AsymptoticBesselI[n + 5/2, σ, order + 1])
          , {κa, ∞, order + 1}]] /. {κa → κ a}
       , {s, {1, -1}}]]
   ];
  SFTasymptotic[poo, pyy, θθ, bb, cc, κκ, aa, nx, ny, nz, order]
 ]
End[];
```

SFTrestricted

End of package

```
EndPackage[]
```