RB-SFA: High Harmonic Generation in the Strong Field Approximation via *Mathematica*

Readme

RB-SFA is a compact and flexible *Mathematica* package for calculating High Harmonic Generation emission within the Strong Field Approximation. It combines *Mathematica*'s analytical integration capabilities with its numerical calculation capacities to offer a fast and user-friendly plug-and-play solver for calculating harmonic spectra and other properties. In addition, it can calculate first-order nondipole corrections to the SFA results to evaluate the effect of the driving laser's magnetic field on harmonic spectra.

The name RB-SFA comes from its first application (as Rotating Bicircular High Harmonic Generation in the Strong field Approximation) but the code is general so RB-SFA just stands for itself now. This first application was used to calculate the polarization properties of the harmonics produced by multi-colour circularly polarized fields, as reported in the paper

Spin conservation in high-order-harmonic generation using bicircular fields. E. Pisanty, S. Sukiasyan and M. Ivanov. *Phys. Rev. A* **90**, 043829 (2014), arXiv:1404.6242.

This code is dual-licensed under the GPL and CC-BY-SA licenses. If you use this code or its results in an academic publication, please cite the paper above or the GitHub repository where the latest version will always be available. An example citation is

E. Pisanty. RB-SFA: High Harmonic Generation in the Strong Field Approximation via *Mathematica*. https://github.com/episanty/RB-SFA (2016).

This software consists of the notebook RB-SFA.nb, which contains the implementation and generates the package file RB-SFA.m, this Usage and Examples notebook, which explains how to use the code and documents the calculations used in the original publication, a Quantum Orbits Usage notebook that works through the (experimental) quantum-orbits functionality, and the subpackage RootFinder.

Specifications

This code calculates the harmonic dipole as per the original Lewenstein *et al.* paper,

M. Lewenstein, Ph. Balcou, M.Yu. Ivanov, A. L'Huillier and P.B. Corkum. Theory of high-harmonic generation by low-frequency laser fields. *Phys. Rev. A* **49** no. 3, 2117 (1994).

The time-dependent dipole is given by

$$\boldsymbol{D}(t) = i \int_0^t dt' \int d^3\boldsymbol{p}\, \boldsymbol{d}(\boldsymbol{\pi}(\boldsymbol{p}, t)) \times \boldsymbol{F}(t') \cdot \boldsymbol{d}(\boldsymbol{\pi}(\boldsymbol{p}, t')) \times \exp[-i\, S(\boldsymbol{p}, t, t')] + c.c.$$

Here $\boldsymbol{F}(t)$is the time-dependent applied electric field, with vector potential $\boldsymbol{A}(t)$; the electron charge is taken to be -1. The integration time t' can be interpreted as the ionization time and the integration limit t represents the recollision time. (Upon applying the saddle-point approximation to the temporal integrals these notions apply exactly to the corresponding quantum orbits.) The ionization and recollision events are governed by the corresponding dipole matrix elements, which are taken for a hydrogenic 1s-type orbital with characteristic momentum κ and ionization potential $I_p = \frac{1}{2}\kappa^2$:

$$\boldsymbol{d}(\boldsymbol{p}) = \left\langle \boldsymbol{p} \middle| \hat{\boldsymbol{d}} \middle| g \right\rangle = \frac{8 i}{\pi} \frac{\sqrt{2\kappa^5}\, \boldsymbol{p}}{(\boldsymbol{p}^2+\kappa^2)^3}.$$

(Taken from B. Podolsky & L. Pauling. *Phys. Rev.* **34** no. 1, 109 (1929).) The displaced momentum $\boldsymbol{\pi}(\boldsymbol{p}, t) = \boldsymbol{p} + \boldsymbol{A}(t)$ is used to aid in the generalization to nondipole cases.

The action

$$S(\boldsymbol{p}, t, t') = \int_{t'}^{t} \left(\frac{1}{2}\kappa^2 + \frac{1}{2}\boldsymbol{\pi}(\boldsymbol{p}, \tau)^2 \right) d\tau$$

is the governing factor of the integral. It is highly oscillatory and must be dealt with carefully. The momentum integral is approximated using the saddle point method; this is valid and straightforward since there is always one unique saddle point,

$$\boldsymbol{p}_s(t, t') = -\frac{\int_{t'}^{t} \boldsymbol{A}(t'')\, dt''}{t-t'},$$

as long as the action's dependence on the momentum is quadratic. After the momentum has been integrated via the saddle point method, the time-dependent dipole is given by

$$\boldsymbol{D}(t) = i \int_0^t dt' \left(\frac{2\pi}{\epsilon + i(t-t')} \right)^{3/2} \boldsymbol{d}(\boldsymbol{\pi}(\boldsymbol{p}_s(t, t'), t)) \times \boldsymbol{F}(t') \cdot \boldsymbol{d}(\boldsymbol{\pi}(\boldsymbol{p}_s(t, t'), t')) \times \exp[-i\, S(\boldsymbol{p}_s(t, t'), t, t')] + c.c.$$

A regularization factor ϵ has been introduced to prevent divergence of the prefactor as t' approaches t, representing the fact that at $t = t'$ the $\boldsymbol{p}$ integral no longer has a quadratic exponent.

The integral is performed with respect to the excursion time $\tau = t - t'$ for simplicity. To reduce integration time, a gating function gate$[\omega\tau]$ has been introduced, and the integration time has been cut short to a controllable length. This is physically reasonable as the wavepacket spreading (which scales as $\tau^{-3/2}$) makes the contributions after the gate negligible.

More physically, suppressing the contributions from large excursion times represents the fact that the extra-long trajectories they represent produce harmonics which are much harder to phase match (as their harmonic phase is very sensitively dependent on many factors) which means that they are rarely present in experimental spectra unless a concerted effort is performed to observe them.

First-order nondipole calculations

This code can also be used to calculate the first-order effect of nondipole terms, which become relevant at high intensities or long wavelengths, when the electron's speed from the laser-driven oscillations (which scales as F/ω) becomes comparable to the speed of light (equal to $c = 1/\alpha \approx 137$ in atomic units). (More specifically, the nondipole effects on the harmonic generation become relevant when the magnetic pushing per cycle, which offsets the returning wavepacket by $F^2/c\omega^3$, becomes comparable with the wavepacket spread upon recollision.)

This code implements a generalization of the theory presented in various forms in the papers N.J. Kylstra et al. *J. Phys B: At. Mol. Opt. Phys.* **34** no. 3, L55 (2001); N..J. Kylstra et al. *Laser Phys.* **12** no. 2, 409 (2002); M.W. Walser et al. *Phys. Rev. Lett.* **85** no. 24, 5082 (2000); V. Averbukh et al. *Phys Rev. A* **65,** 063402 (2002); and C.C. Chirilă et al. *Phys. Rev. A* **66,** 063411 (2002).

The nondipole-HHG code calculates the harmonic dipole of exactly the same form as the standard SFA,

$$\boldsymbol{D}(t) = i \int_0^t dt' \int d^3\boldsymbol{p}\, \boldsymbol{d}(\boldsymbol{\pi}(\boldsymbol{p}, t, t')) \times \boldsymbol{F}(t') \cdot \boldsymbol{d}(\boldsymbol{\pi}(\boldsymbol{p}, t', t')) \times \exp[-i\, S(\boldsymbol{p}, t, t')] + c.c.$$

with $S(\boldsymbol{p}, t, t') = \int_{t'}^{t} dt'' \left(\frac{1}{2}\kappa^2 + \frac{1}{2}\boldsymbol{\pi}(\boldsymbol{p}, t'', t')^2\right)$ as before, but with a nondipole term in the displaced momentum, which now reads

$$\boldsymbol{\pi}(\boldsymbol{p}, t, t') = \boldsymbol{p} + \boldsymbol{A}(t) - \int_{t'}^{t} \nabla \boldsymbol{A}(\tau) \cdot (\boldsymbol{p} + \boldsymbol{A}(\tau))\, d\tau,$$

or in component notation

$$\pi_j(\boldsymbol{p}, t, t') = p_j + A_j(t) - \int_{t'}^{t} \partial_j A_k(\tau)\, (p_k + A_k(\tau))\, d\tau.$$

Usage and Examples

Loading the package

You can use this software

- within the RB-SFA notebook itself by simply running the initialization cells of that notebook, or
- from an external notebook by loading it as a package.

In the latter case, place a copy of the package file RB-SFA.m on the same directory as your notebook and run the loading command

```
Needs["RBSFA`", FileNameJoin[{NotebookDirectory[], "RB-SFA.m "}]]
```

You can also call the package from another directory by suitably modifying the directory call. If you plan on using this package in the long term you can use the File > Install prompt, in which case the package is simply loaded as Needs["RBSFA`"], though this is not particularly recommended. (A better choice is to include a soft link called RBSFA.m in your $UserBaseDirectory/Applications/ directory to the file RB-SFA.m. This works just fine and is easy to undo if required.)

To print the version of the package in use, use the command

```
$RBSFAversion
```

```
RB-SFA v2.1.1, Wed 8 Jun 2016 15:00:40
```

There are also commands to get the $RBSFAtimestamp directly, as well as the git $RBSFAcommit hash and message.

Simple usage

For basic usage, simply call the main numerical integrator, makeDipoleList, with the vector potential you want to use, and provide any parameters you wish to specify using the FieldParameters option.

```
AbsoluteTiming[
  simpleDipole = makeDipoleList[VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}],
       FieldParameters → {F → 0.05, ω → 0.057}];
]
```

```
{3.208, Null}
```

Calling the function with insufficient parameters will produce error messages:

```
makeDipoleList[VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}]]
```

makeDipoleList::pot :

The vector potential A provided as VectorPotential→Function[t, {F Sin[ω t]/ω, 0, 0}] is incorrect or is missing FieldParameters.

Its usage as A[4.989144044218128`] returns {F Sin[4.98914 ω]/ω, 0, 0} and should return a list of numbers.

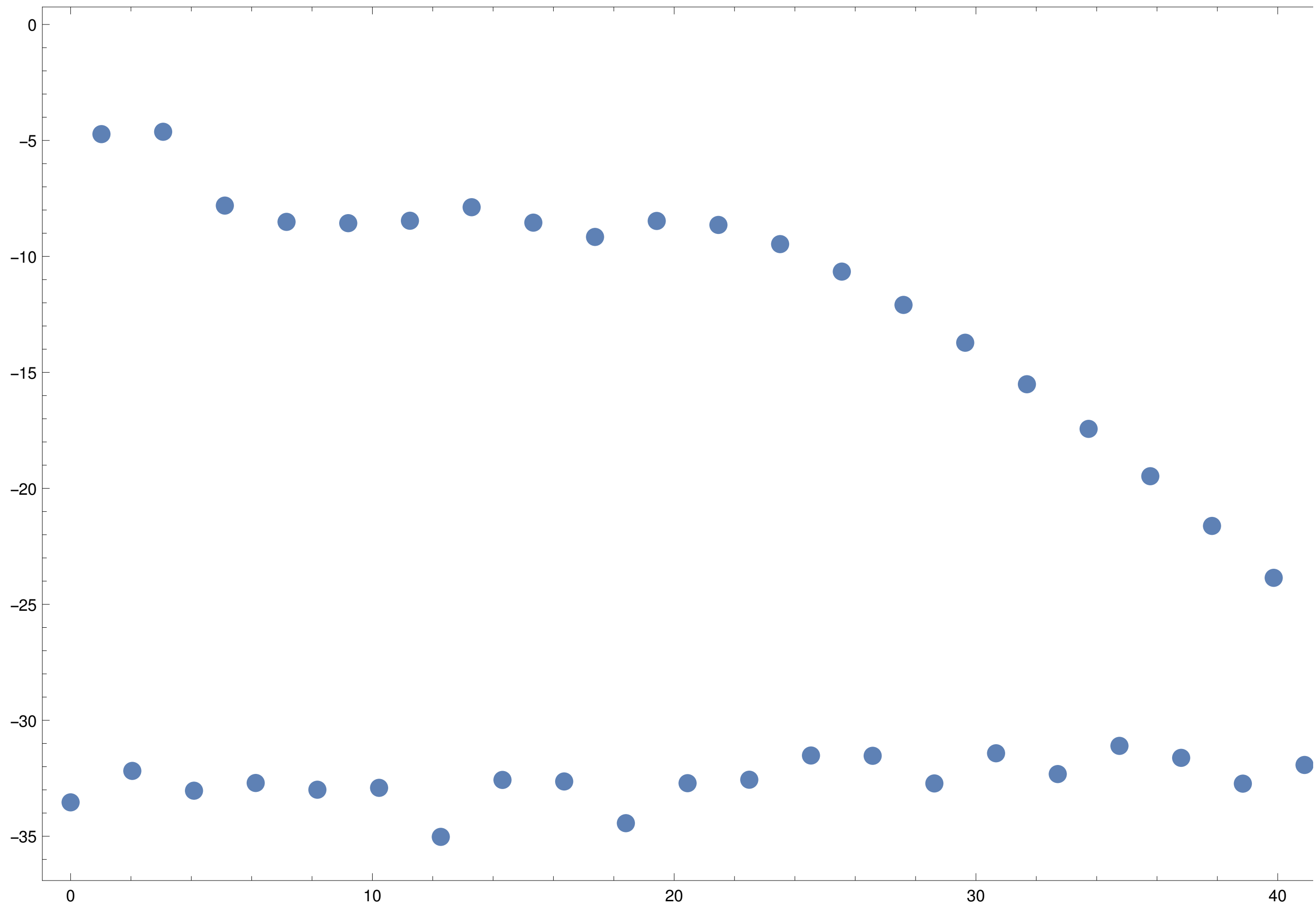
```
$Aborted
```

The symbol ω is taken to be the carrier frequency, and is set by default to $\omega = 0.057$ atomic units, corresponding to a wavelength of 800 nm. If the carrier frequency is changed, this must be specified on **both** the field parameters and the explicit option for the integrator, as

```
makeDipoleList[VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}],
  FieldParameters →{F→0.05, ω→0.0456}, CarrierFrequency→0.0456]
```

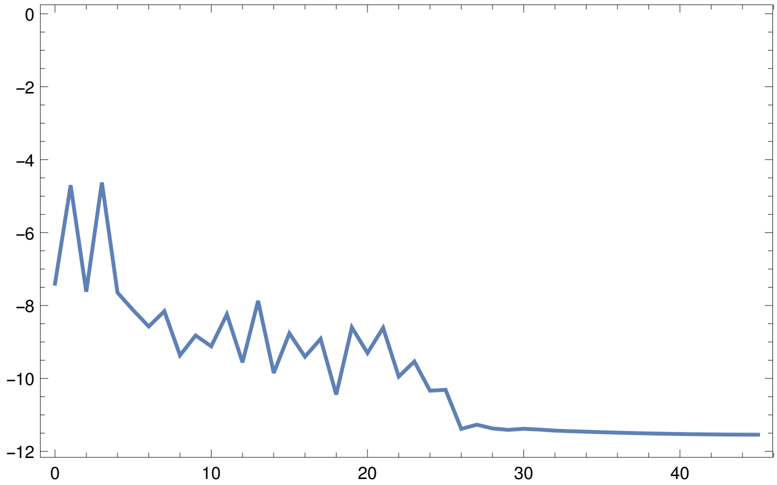
To see the spectrum, use the getSpectrum and the spectrumPlotter commands, such as

```
spectrumPlotter[getSpectrum[Most[simpleDipole]], Joined→False]
```



Note here the use of Most on the dipole when a monochromatic field is indicated. This ensures that the signal is actually periodic (i.e. it eliminates repetition between the initial and final points, which are separated by exactly one period). If this is not done, the spectrum is much noisier:

```
spectrumPlotter [getSpectrum [simpleDipole ], ImageSize → 400]
```

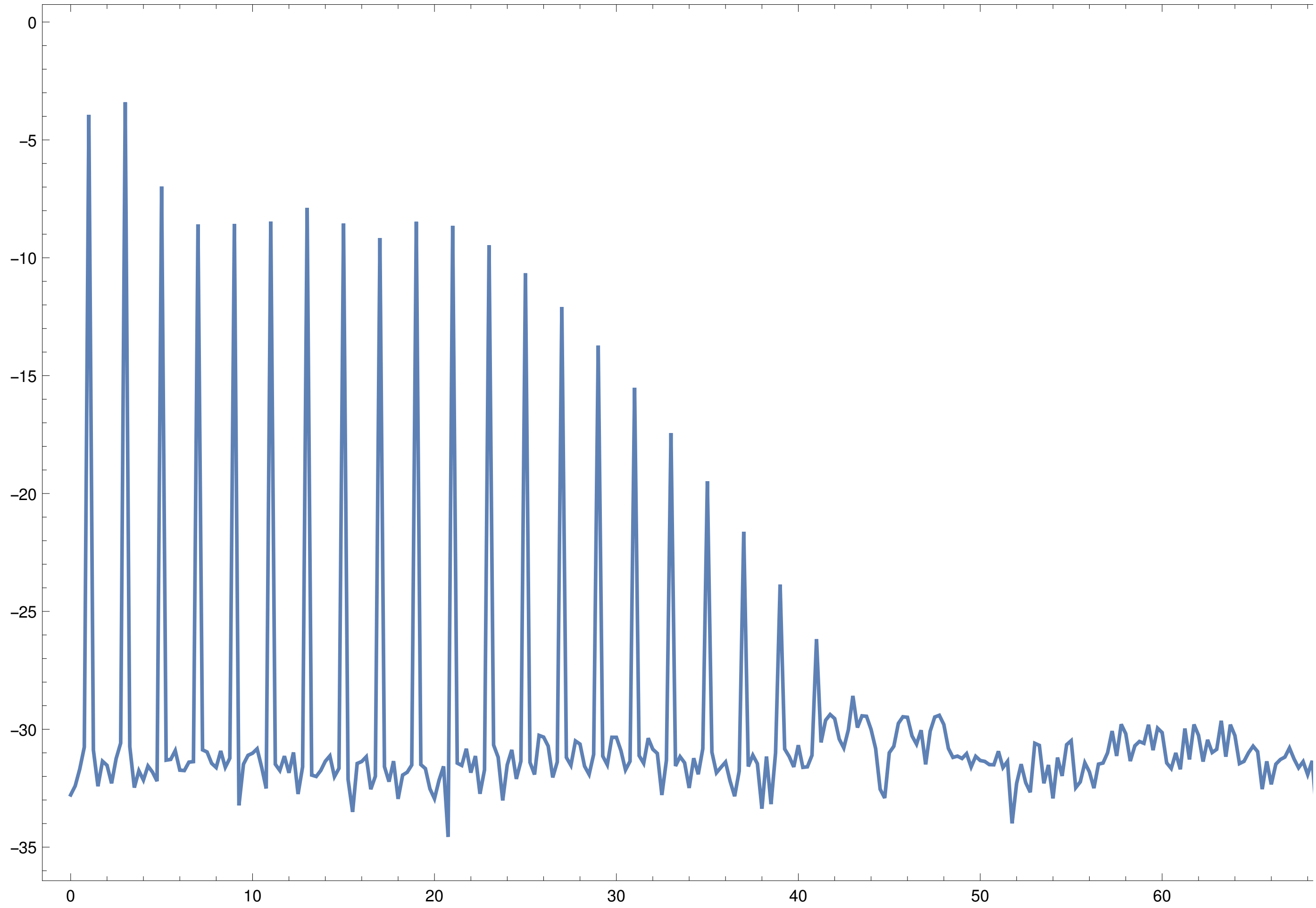


The default options are built for a periodic pulse for which simple functions of the vector potential can be integrated analytically, and for which only a single period of integration is necessary. More periods can be specified using the TotalCycles option. Similarly, the PointsPerCycle option controls the number of points per period.

```
AbsoluteTiming [
  biggerDipole= makeDipoleList [VectorPotential→ Function[t, {F/ω Sin[ω t], 0, 0}],
        FieldParameters → {F → 0.05, ω → 0.057}, TotalCycles → 4, PointsPerCycle→ 150];
]
{33.8262, Null}
```

To get a correct spectrum plot, give these settings to the spectrum plotter.

```
spectrumPlotter [getSpectrum [Most[biggerDipole]], TotalCycles→4, PointsPerCycle→150]
```



You can specify a Target chemical species using the option

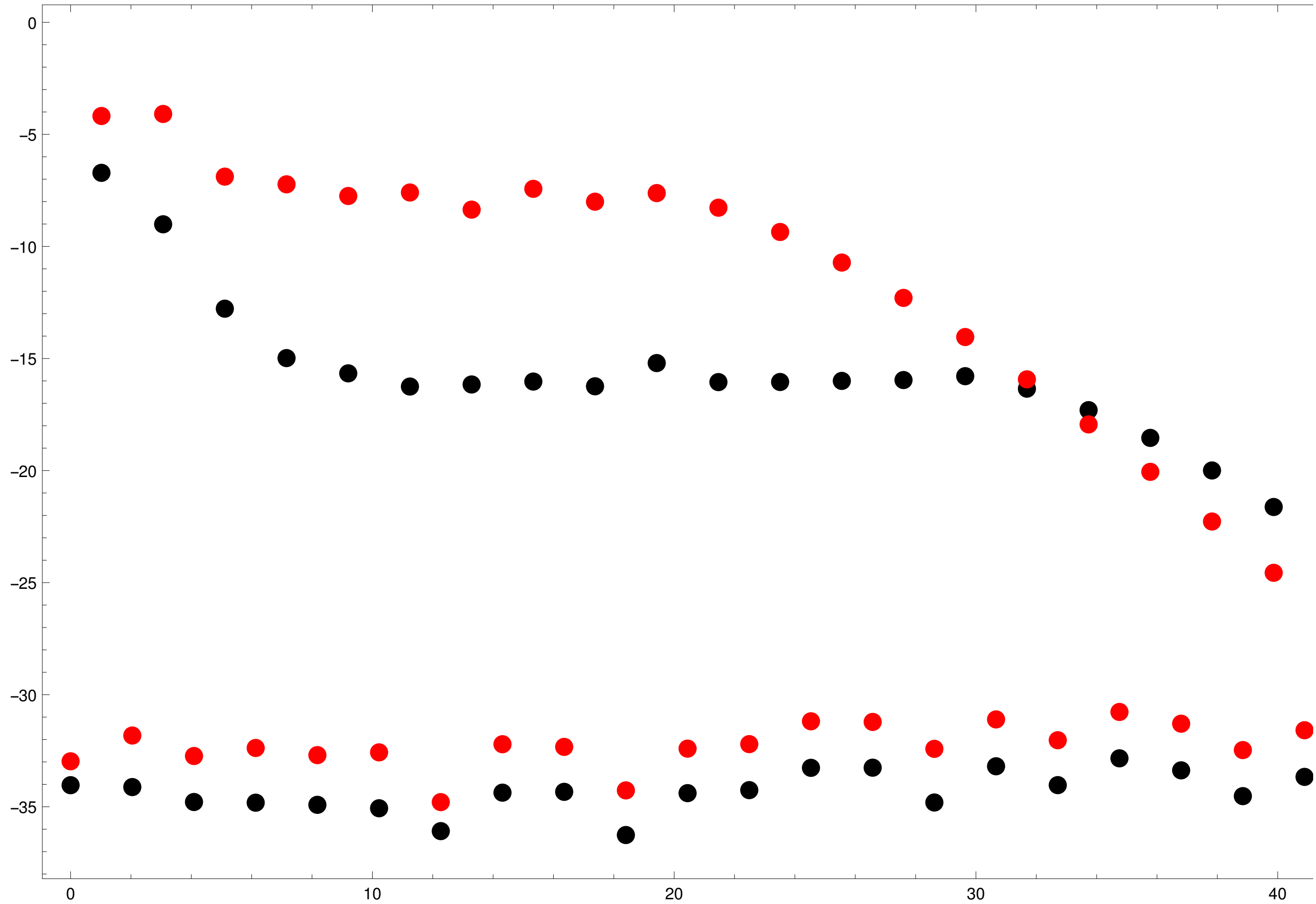
```
?Target
```

Target is an option for makeDipoleList which specifies chemical species producing the HHG emission, pulling the ionization potential from the Wolfram ElementData curated data set.

i.e. using the syntax

```
AbsoluteTiming[
  heliumDipole = makeDipoleList[VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}],
      FieldParameters → {F→0.05, ω→0.057}, Target→"Helium "];
  xenonDipole= makeDipoleList[VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}],
      FieldParameters → {F→0.05, ω→0.057}, Target→"Xenon"];
]
{9.17929, Null}
```

```
Show[{
    spectrumPlotter [getSpectrum [Most[heliumDipole ]], Joined→ False, PlotStyle→Black],
    spectrumPlotter [getSpectrum [Most[xenonDipole]], Joined→ False, PlotStyle→Red]
  }]
```



For convenience, the function getIonizationPotential gives a public-facing access to this functionality, via

```
?getIonizationPotential
```

getIonizationPotential [Target] returns the ionization potential of an atomic target, e.g. "Hydrogen", in atomic units.

getIonizationPotential [Target,q] returns the ionization potential of the q–th ion of the specified Target, in atomic units.

so that e.g.

```
{"H", #, UnitConvert[Quantity[#, "Hartrees"], "Electronvolts"]} &[
  getIonizationPotential["Hydrogen"]]
{"He⁺", #, UnitConvert[Quantity[#, "Hartrees"], "Electronvolts"]} &[
  getIonizationPotential["Helium ", 1]]
```

```
{H, 0.49971, 13.598 eV }
```

```
{He⁺, 1.9998, 54.418 eV }
```

An ionization potential can also be specified directly:

```
?IonizationPotential
```

"IonizationPotential is an option for makeDipoleList which specifies the ionization potential I_p of the target."

To see the available options for this function (and others), use

```
Options[makeDipoleList]
```

```
{PointsPerCycle→90, TotalCycles→1, CarrierFrequency→0.057, VectorPotential→Automatic,
  FieldParameters →{}, VectorPotentialGradient→None, Preintegrals→Analytic,
  ReportingFunction→Identity, Gate→SineSquaredGate[1/2], nGate→3/2, ϵCorrection→0.1,
  IonizationPotential→0.5, Target→Automatic, DipoleTransitionMatrixElement→hydrogenicDTME,
  PointNumberCorrection →0, Verbose→0, IntegrationPointsPerCycle→Automatic}
```

All options have suitable information messages.

```
?VectorPotential
```

VectorPotential is an option for makeDipole list which specifies the field 's vector potential . Usage should be VectorPotential→A, where A[t]//.pars must yield a list of numbers for numeric t and parameters indicated by FieldParameters →pars.

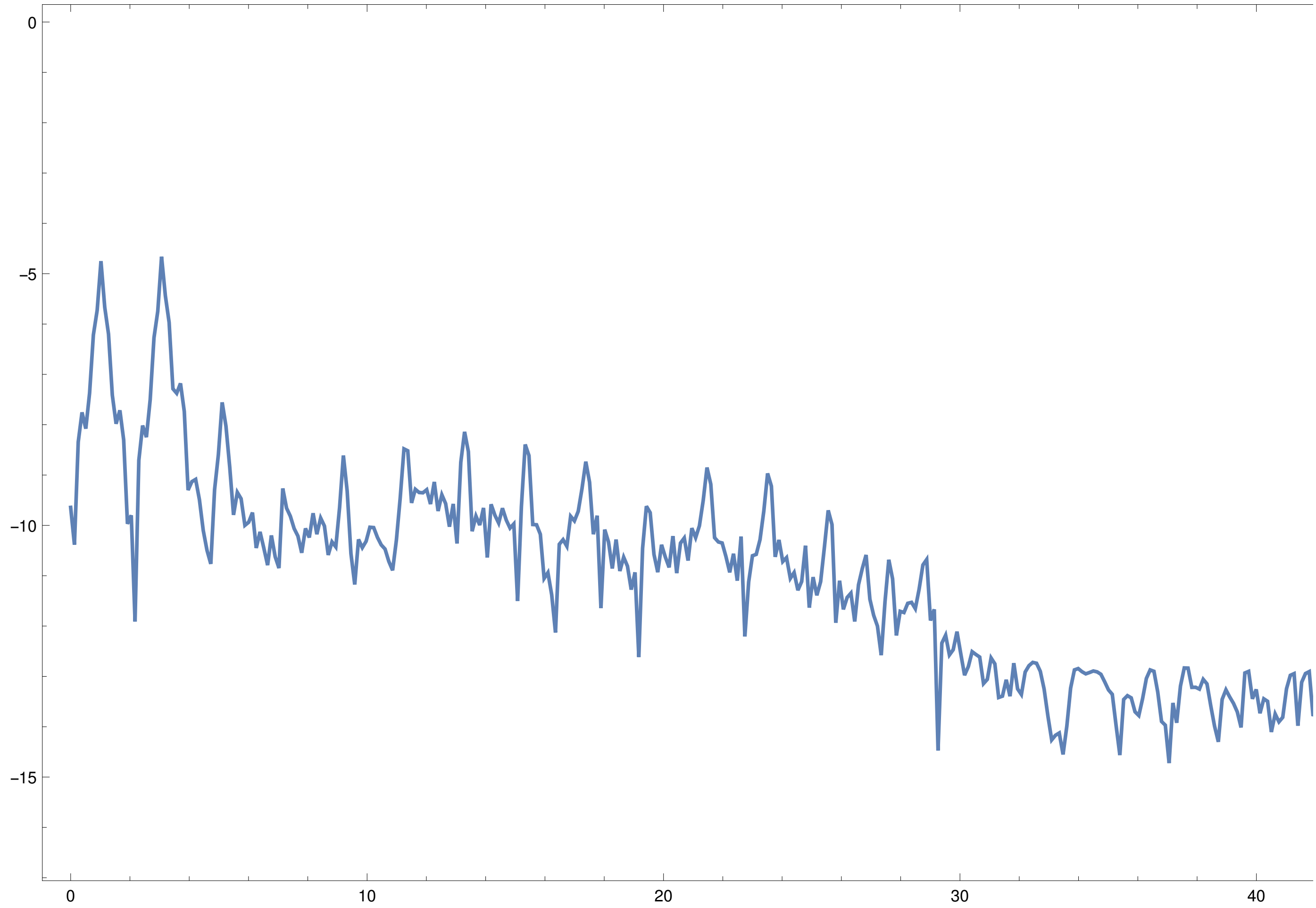
Using numerical integration for the preintegrals

Dipole case

To simulate a pulse with an envelope, it can be convenient to perform the preintegrals numerically, using the option Preintegrals→”Numeric”. These cases are generally slower but mainly because they require many more periods of integration.

```
AbsoluteTiming[
  numericallyIntegratedDipole =
      makeDipoleList[VectorPotential→Function[t, {F/ω envelope[t] Sin[ω t], 0, 0}]
        , FieldParameters →{ω→0.057, F→0.055, envelope→cosPowerFlatTop[0.057, 8, 16]}
        , TotalCycles→8
        , Preintegrals→"Numeric "
      ];
]
{27.5127, Null}
```

```
spectrumPlotter [getSpectrum [numericallyIntegratedDipole ]]
```



When using flat top pulses, and other waveforms that depend on Piecewise functions, it is possible that the function will return errors caused by an Indeterminate derivative being evaluated at the corners of the envelope.

```
AbsoluteTiming [
  flatTopPulseDipole=
      makeDipoleList [VectorPotential→Function[t, {F/ω envelope[t] Sin[ω t], 0, 0}],
        FieldParameters →{ω→0.057, F→0.055, envelope→flatTopEnvelope[0.057, 8, 2]},
        TotalCycles→8, Preintegrals→"Numeric "];
]
{29.4577, Null}
```

In these cases, use a numeric test to diagnose what’s happened

```
Tally[flatTopPulseDipole/. _?NumberQ  →✓]
{{{✓, ✓, ✓}, 721}}
```

and if the function is returning non-numeric values, it can help to fiddle with the PointNumberCorrection option.

```
?PointNumberCorrection
```

PointNumberCorrection is an option for makeDipoleList and timeAxis which specifies an extra number of points to be integrated over, which is useful to prevent Indeterminate errors when a Piecewise envelope is being differentiated at the boundaries.

Nondipole case

The numerical Preintegrals can be used in the nondipole case but they're obviously much slower. The number of preintegrals to find numerically increases from two in the dipole case ($\int \boldsymbol{A}(\tau)\, d\tau$ and $\int \boldsymbol{A}(\tau)^2\, d\tau$) to eight with the nondipole contributions, three of them parametrized by t'. The main load, however, is not in numerically calculating these integrals via NDSolve constructs, but rather in the added strain of accessing the preintegrals as InterpolatingFunction objects once they've been calculated, from the main integration loop.

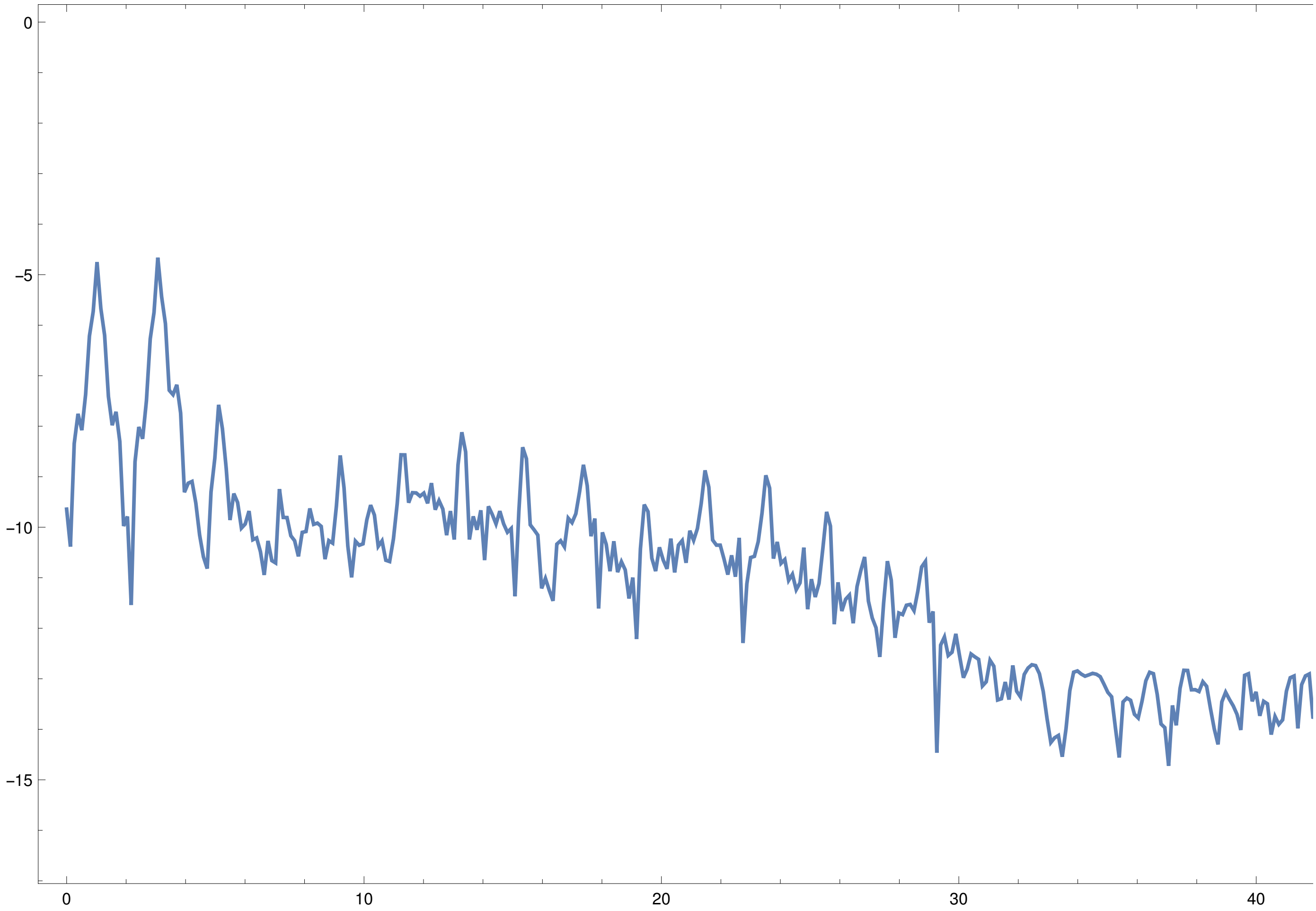
The numerical preintegrals for the nondipole case should currently be considered experimental.

```
DateString[]
AbsoluteTiming[
  numericallyIntegratedNonDipoleCaseDipole = makeDipoleList[
        VectorPotential → Function[t, {F/ω envelope[t] Sin[ω t], 0, 0}],
        VectorPotentialGradient →
          Function[t, {{0, 0, 0}, {0, 0, 0}, {-(k F)/ω envelope[t] Sin[ω t], 0, 0}}],
        FieldParameters → {ω → 0.057, F → 0.055,
            envelope → cosPowerFlatTop[0.057, 8, 16], k → ωω, α → 1/20}
        , TotalCycles → 8, Preintegrals → "Numeric "
      ];
]
DateString[]
Beep[]
```

```
Fri 13 May 2016 17:20:24

{146.097, Null}

Fri 13 May 2016 17:22:50
```

This requires some attention - currently taking too long to calculate.

```
spectrumPlotter [getSpectrum [numericallyIntegratedNonDipoleCaseDipole]]
```



Parallelization

Parallelizing single instances

For faster evaluation of a single instance, it is possible to parallelize the evaluation, by adding the option RunInParallel → True.

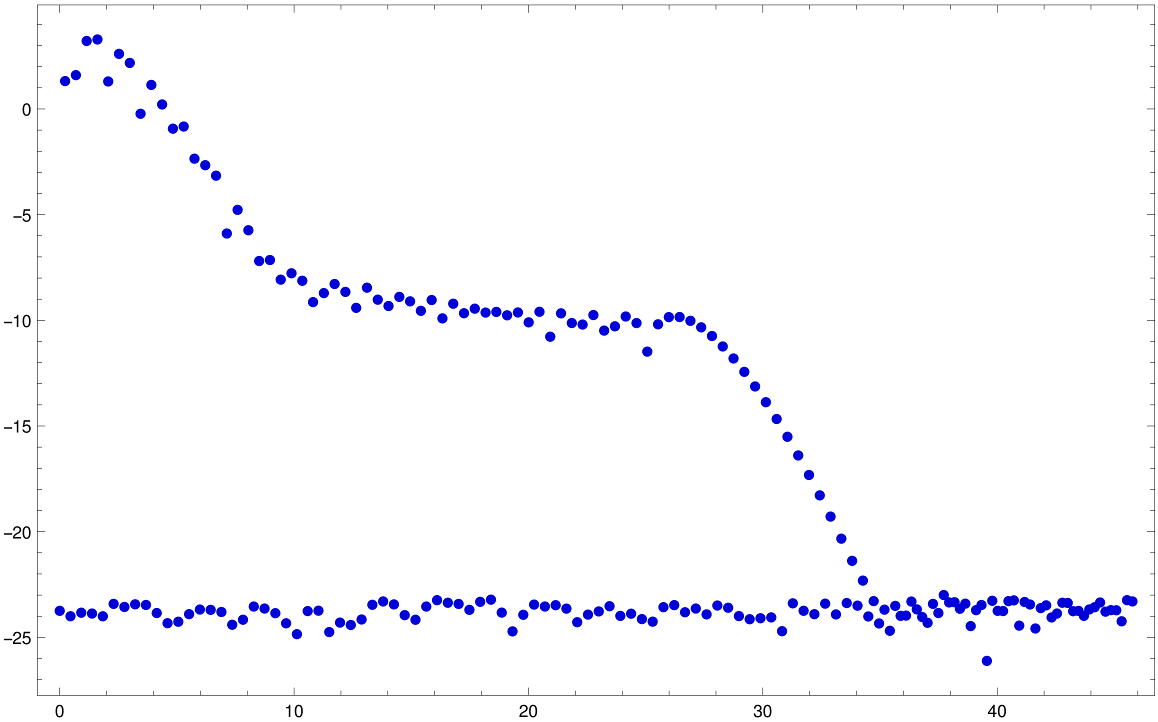
```
AbsoluteTiming [directDipole=
    makeDipoleList [VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}], FieldParameters →
      {F→√10 0.05, ω→0.057}, PointsPerCycle→400, RunInParallel→False];]
AbsoluteTiming [parallelizedDipole=makeDipoleList [
    VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}],
    FieldParameters →{F→√10 0.05, ω→0.057}, PointsPerCycle→400, RunInParallel→True];]
```

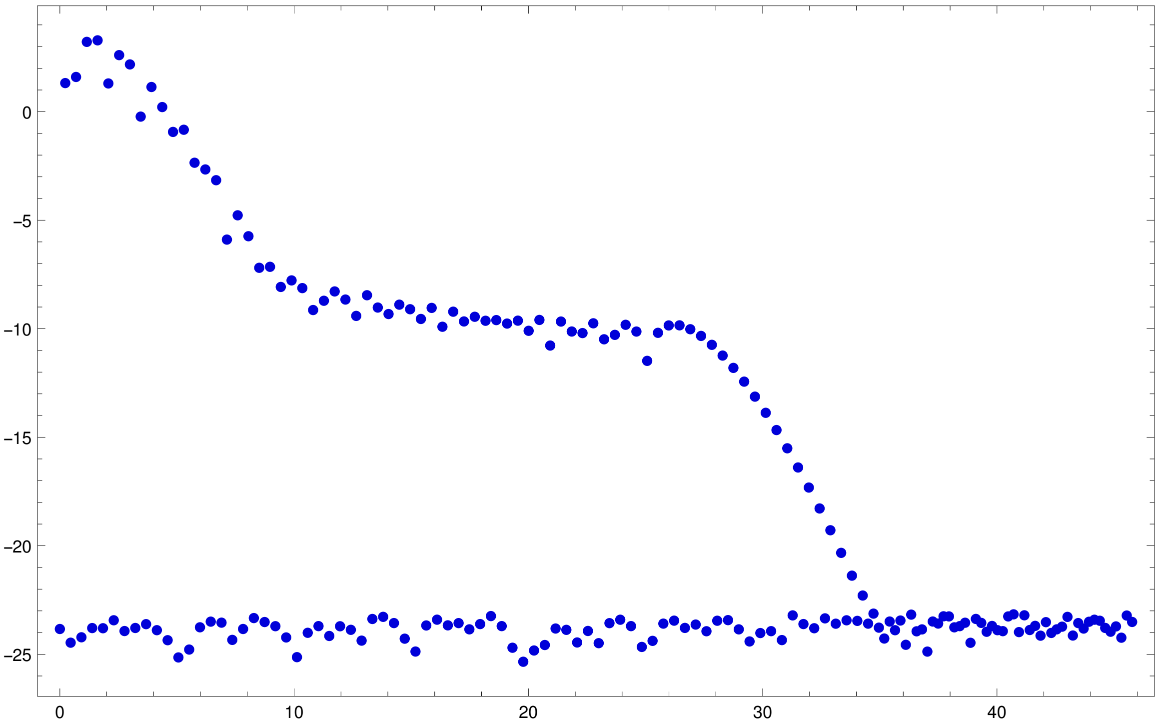
```
{50.7121, Null}
```

```
{52.8681, Null}
```

NOTE that this requires further attention - it appears that the parallelization is not working at the moment.

```
Row[{
    spectrumPlotter [getSpectrum [directDipole[[1 ;; -2]]],
      Joined→False, PlotStyle→Darker[Blue, 0.15], ImageSize →600],
    spectrumPlotter [getSpectrum [parallelizedDipole[[1 ;; -2]]],
      Joined→False, PlotStyle→Darker[Blue, 0.15], ImageSize →600]
  }]
```





Unfortunately, the in-package single-instance parallelization can be unstable on occasion; this is probably due to a bug in ParallelTable (which can, under enough load, return different results to Table, in which case results typically differ run-to-run) that has proven so far very difficult to diagnose.

In such cases, the RunInParallel option takes a third possibility - an explicit set of commands, {TableCommand, SumCommand}, to use in the iteration.

```
?RunInParallel
```

> RunInParallel is an option for makeDipoleList which controls whether each RB–SFA instance is parallelized. It accepts False as the (Automatic) option, True, to parallelize each instance, or a pair of functions {TableCommand, SumCommand} to use for the iteration and summing, which could be e.g. {Inactive[ParallelTable], Inactive[Sum]}.

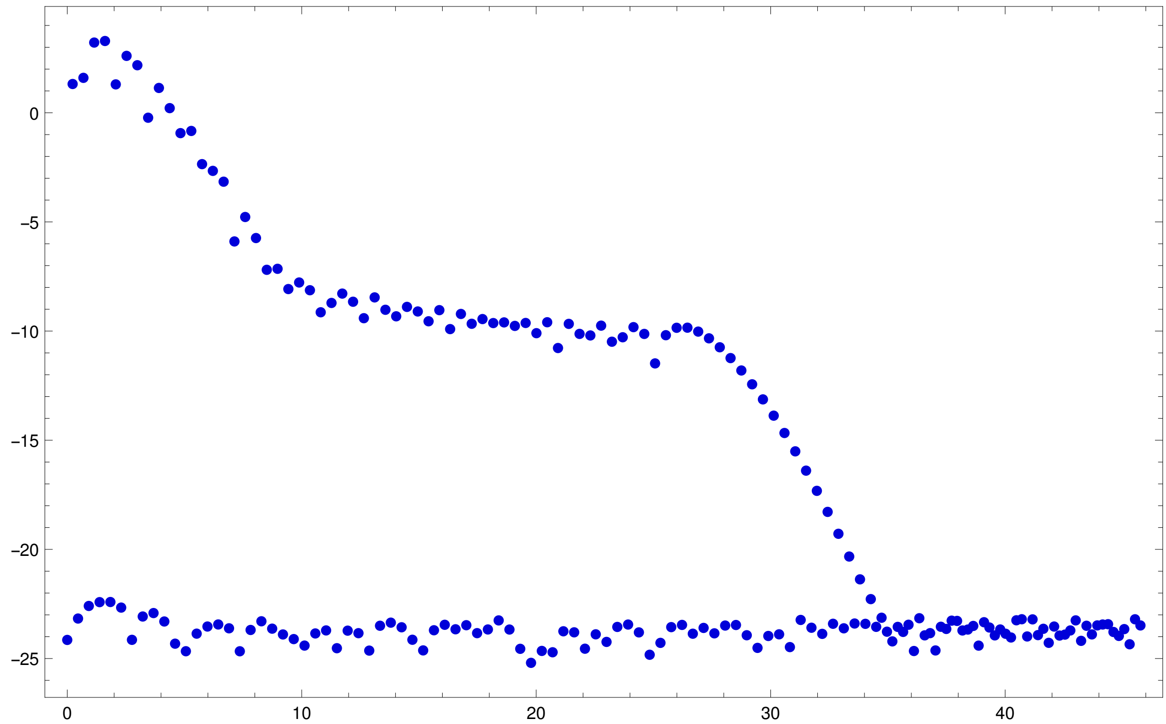
This is meant to be used by changing those commands to dud versions which can be sprung up later. The ideal use case (in v10 and up) is via Inactive commands, which return as

```
makeDipoleList[VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}],
  FieldParameters → {F → √10 0.05, ω → 0.057}, Gate → (1 &), PointsPerCycle → 400
  , RunInParallel → {Inactive[ParallelTable], Inactive[Sum]}
 ] /. {RBSFA`Private`t → t, RBSFA`Private`τ → τ}
```

```
ParallelTable[0.275578 Sum[{((56.7185 + 0. ⅈ)
      ⅇ^(-ⅈ (-(-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)])^2/((0.-0.1 ⅈ)+τ) + 1/2 τ (1. + (-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)])^2/(((0.-0.1 ⅈ)+τ)^2)) + 1/2 (7.69468 (0.5 t-4.38596 Sin
      (1/(0.1 + ⅈ τ))^(3/2)
      (-((-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)]) / ((0. - 0.1 ⅈ) + τ)) +
         2.77393 Sin[0.057 t]) (0. - ((0. + 0.56941 ⅈ) Cos[0.057 (t-τ)]
            (-((-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)]) / ((0. - 0.1 ⅈ) + τ)) +
               2.77393 Sin[0.057 (t-τ)])) /
          (1. + (-((-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)]) / ((0. - 0.1 ⅈ) + τ)) +
               2.77393 Sin[0.057 (t-τ)])^2)^3)) /
    (1. + (-((-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)]) / ((0. - 0.1 ⅈ) + τ)) +
         2.77393 Sin[0.057 t])^2)^3,
   (0. + 0. ⅈ) (1/(0.1 + ⅈ τ))^(3/2) (0. - ((0. + 0.56941 ⅈ) Cos[0.057 (t-τ)]
        (-((-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)]) / ((0. - 0.1 ⅈ) + τ)) +
           2.77393 Sin[0.057 (t-τ)])) /
      (1. + (-((-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)]) / ((0. - 0.1 ⅈ) + τ)) +
           2.77393 Sin[0.057 (t-τ)])^2)^3),
   (0. + 0. ⅈ) (1/(0.1 + ⅈ τ))^(3/2) (0. - ((0. + 0.56941 ⅈ) Cos[0.057 (t-τ)]
        (-((-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)]) / ((0. - 0.1 ⅈ) + τ)) +
           2.77393 Sin[0.057 (t-τ)])) /
      (1. + (-((-48.6654 Cos[0.057 t] + 48.6654 Cos[0.057 (t-τ)]) / ((0. - 0.1 ⅈ) + τ)) +
           2.77393 Sin[0.057 (t-τ)])^2)^3)},
  {τ, 0, 165.347, 0.275578}], {t, 0, 110.231, 0.275578}]
```

and which can then be sprung into action using Activate:

```
AbsoluteTiming[postActivatedDipole = Activate[
    makeDipoleList[VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}],
      FieldParameters → {F → √10 0.05, ω → 0.057}, PointsPerCycle → 400,
      RunInParallel → {Inactive[ParallelTable], Inactive[Sum]}]
   ];]
spectrumPlotter[getSpectrum[postActivatedDipole[[1 ;; -2]]],
  Joined → False, PlotStyle → Darker[Blue, 0.15], ImageSize → 600]
{6.14601, Null}
```



This looks like it shouldn't change anything, but it can help fix a noisy ParallelTable output. It can also, inexplicably, be rather faster than the in-package parallelization. (For a cleaner example of the latter, see this mathematica.stackexchange question.)

Multiple instances in parallel

Alternatively, one can also parallelize over each run by using ParallelTable and similar commands. In general, this requires some careful handling of contexts; in this package this has been resolved by calling

```
DistributeDefinitions["RBSFA`"];
```

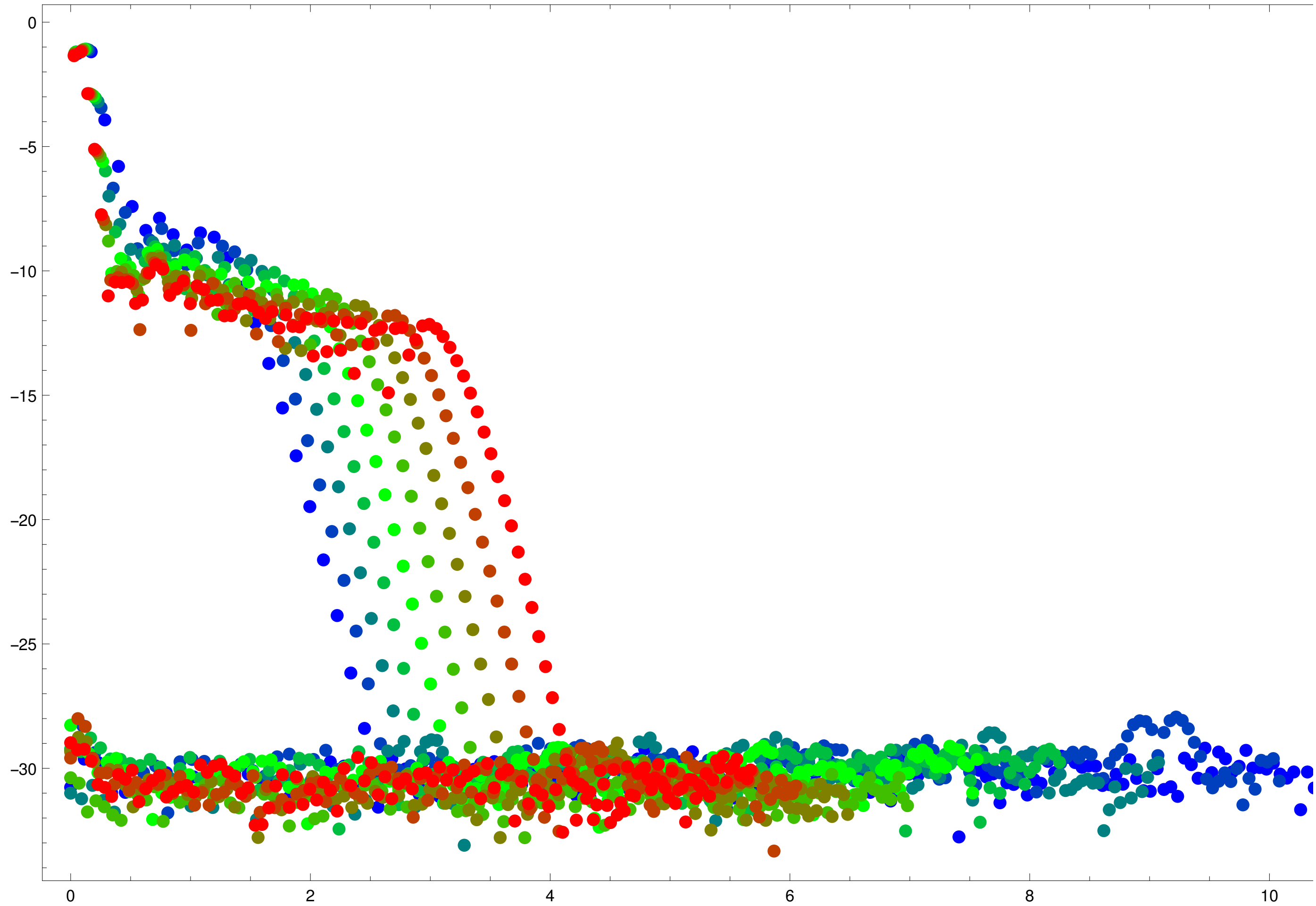
at the end of the package, which should pre-load all the RB-SFA definitions into each subkernel as it is launched, even if a kernel dies or is re-spawned. If parallelization seems not to be working (calculations taking too long and using a single core), there is a chance that the parallel kernels are returning the RBSFA` functions unevaluated to the main kernel, which is then doing the work. This can be diagnosed by inserting an appropriate print statement of the $KernelId to force the calculating kernel to identify itself (0 for the main kernel, >0 for subkernels), and it can be fixed by suitable use of the DistributeDefinitions function and the $DistributedContexts variable.

In addition to this, if a variable or function is used to store the results, this must be synchronized using SetSharedFunction or SetSharedVariable, as usual.

```
SetSharedFunction[wavelengthScanDipole];

ParallelTable[
  Print[AbsoluteTiming [
      wavelengthScanDipole[λ] =
          makeDipoleList [VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}], FieldParameters →
              {F→0.05, ω→45.6/λ}, CarrierFrequency→45.6/λ, PointsPerCycle→400];
    ]]
  , {λ, 800, 1600, 100}]
```

```
{58.5366, Null}
{59.3303, Null}
{60.1489, Null}
{61.4793, Null}
{60.5053, Null}
{61.4832, Null}
{61.1334, Null}
{61.0625, Null}
{52.589, Null}
{Null, Null, Null, Null, Null, Null, Null, Null, Null}
```

```
Show[Table[
    spectrumPlotter [getSpectrum [Most[wavelengthScanDipole[λ]]],
      PlotStyle→Blend[{Blue, Green, Red}, λ/800-1], CarrierFrequency→45.6/λ,
      Joined→False, FrequencyAxis→"Frequency", PointsPerCycle→400]
    , {λ, 800, 1600, 100}]]
```



Writing output to file

For very large calculations (many integration points per cycle, in particular), the limiting factor is available memory. In these situations, it can help to write the data directly to a file on disk. This is slower (by a factor of about 2) but it has a roughly constant RAM footprint, so it enables calculations of a bigger size than would be possible otherwise. (Of course, this can also be done from non-parallelized calls!) This is done via the ReportingFunction option:

```
?ReportingFunction
```

ReportingFunction is an option for makeDipole list which specifies a function used to report the results, either internally (by the default, Identity) or to an external file.

In essence, the integration loop consists of a Table construct, which goes over the time *t* at which the integral is performed, and an inner integration construct. Setting an option ReportingFunction→f interposes the function f between these two steps, as

```
Table[ f[ integrator[t] ] , {t, tInitial, tFinal}]
```

The default is f=Identity, which returns its input untouched, but it can also be replaced by a Write construct that can shunt its input to the hard disk without telling the kernel what it is, so it is not kept in memory.

```
Quit

directory = NotebookDirectory[];
filename[F_] :=
   FileNameJoin[{directory, "Field scan data at F=" <> ToString[F] <> ".txt"}];

ParallelTable[
  Print[AbsoluteTiming[
     makeDipoleList[VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}],
        FieldParameters → {ω → 0.057}, CarrierFrequency → 0.057, PointsPerCycle → 400,
        ReportingFunction → Function[Write[filename[F], #]]
      ];
    ]]
  , {F, 0.05, 0.2, 0.025}]

{61.0728, Null}

{61.6888, Null}

{61.9032, Null}

{63.9092, Null}

{54.0083, Null}

{55.1511, Null}

{55.1227, Null}

{Null, Null, Null, Null, Null, Null, Null}
```

The data in the files can then be pulled in quite simply using e.g.

```
Do[ intensityScanDipole[F] = ReadList[filename[F]], {F, 0.05, 0.2, 0.025}]
```

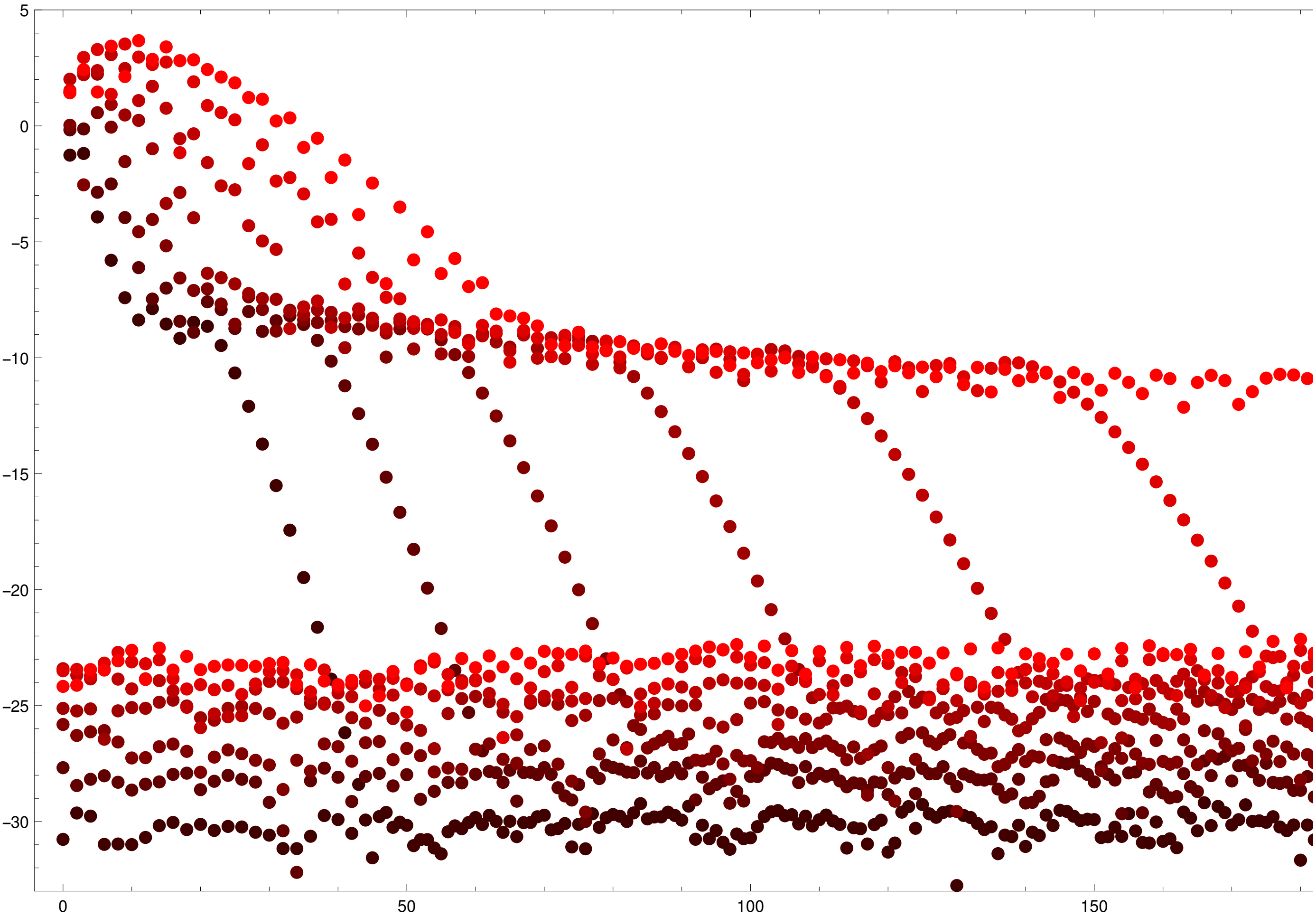
This tends to litter the directories by creating lots of files for different parameters, so it is usually cleaner to Save them into a single file, e.g. using

```
Save[FileNameJoin[{NotebookDirectory[], "Field scan collected data.txt"}],
  intensityScanDipole]
```

which in turn can then be pulled in using

```
<< (FileNameJoin[{NotebookDirectory[], "Field scan collected data.txt"}]);
```

```
Show[Table[
    spectrumPlotter [getSpectrum [Most[intensityScanDipole[F]]],
      CarrierFrequency→0.057, Joined→False, PointsPerCycle→400,
      PlotStyle→Blend[{Black, Red}, F/0.2], PlotRange→{-33, 5}]
    , {F, 0.05, 0.2, 0.025}]]
```



As written, though, this has the disadvantage that each subkernel must access the hard drive for *every* timestep of the computation, which obviously responsible for (at least most of) the slowdown. A middle ground is also possible by choosing an appropriate ReportingFunction: a function which will cache a specific number *k* of results on RAM, and then write them to file all in one go. This is on the development to do (wish) list, and will hopefully be implemented soon - if time allows.

Time and memory use

Benchmark evaluation

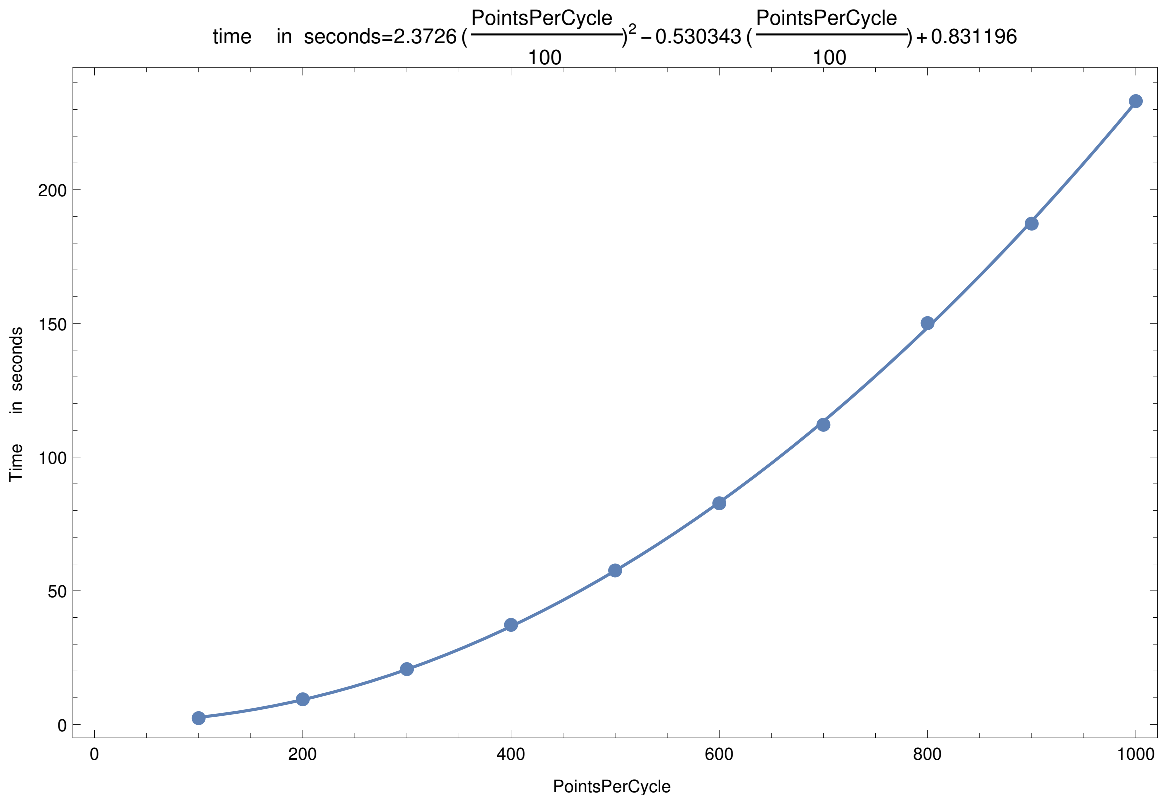
The benchmarks below were taken on a desktop machine with 8-thread, 4-core Intel i7-3770 CPU at 3.40GHz, 16GB RAM, running *Mathematica* 10.0.1 over Ubuntu 14.04. The time taken per computation depends most strongly on the PointsPerCycle used to sample and integrate, and the dependence is therefore quadratic.

```
timingsList = Table[
   {n, AbsoluteTiming [
       MaxMemoryUsed [makeDipoleList [VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}],
           FieldParameters → {F → Sqrt[n/100] 0.053, ω → 0.057}, PointsPerCycle → n]]]}
   ,
   {n,
     100,
     1000,
     100}]
```

```
{{100, {2.36167, 4905296}}, {200, {9.40602, 19018888}},
  {300, {20.6878, 43497000}}, {400, {37.2461, 78976528}}, {500, {57.5847, 118770848}},
  {600, {82.7101, 173233496}}, {700, {112.082, 232621352}}, {800, {150.096, 301125912}},
  {900, {187.298, 387140808}}, {1000, {233.122, 473885368}}}
```

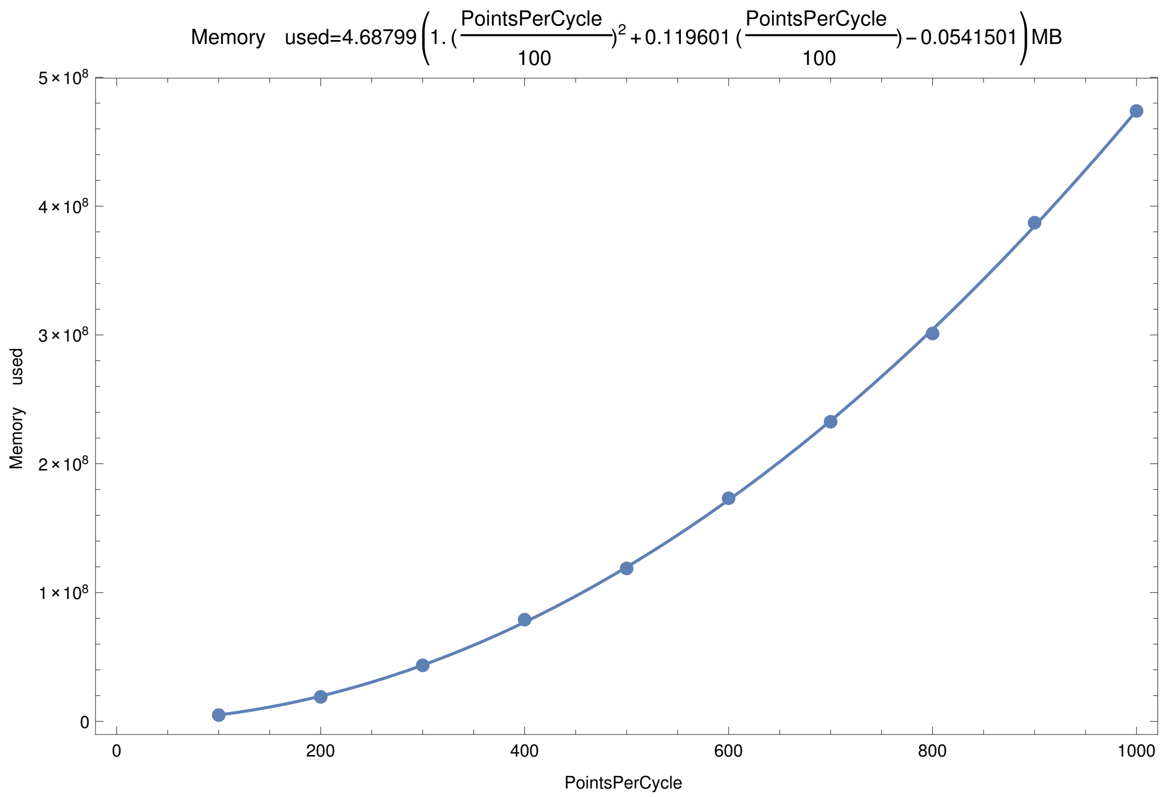
Timings

```
timingsModel = LinearModelFit[(Flatten/@timingsList)[[All, {1, 2}]], {1, n, n^2}, {n}];
Show[{
    ListPlot[
      (Flatten/@timingsList)[[All, {1, 2}]]
    ],
    Plot[timingsModel [n], {n, timingsList[[1, 1]], timingsList[[-1, 1]]}]
  }
  , Frame → True, PlotLabel → Row[{"time  in seconds=", timingsModel [100 "(PointsPerCycle/100)"]}],
  FrameLabel → {"PointsPerCycle", "Time  in seconds"}, ImageSize → 600
]
```



Maximum memory used

```
memoryModel = LinearModelFit[(Flatten/@timingsList)[[All, {1, 3}]], {1, n, n^2}, {n}];
Show[{
    ListPlot[
      (Flatten/@timingsList)[[All, {1, 3}]]
    ],
    Plot[memoryModel[n], {n, timingsList[[1, 1]], timingsList[[-1, 1]]}]
  }
  , Frame → True, PlotLabel →
    Row[{"Memory used=", Simplify[memoryModel[100. "(PointsPerCycle/100)"] 10^-6 "MB"]}],
  FrameLabel → {"PointsPerCycle", "Memory used"}, ImageSize → 600
]
```



In parallel

Inside parallel environments the timings are somewhat slower, by a factor of about 1.8. The timings below were taken with 7 *Mathematica* kernels running in parallel.

```
DistributeDefinitions["RBSFA`"];
```

```
parallelTimingsList = ParallelTable[
   {n, AbsoluteTiming[MaxMemoryUsed[
       makeDipoleList[VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}], FieldParameters →
           {F → Sqrt[n/100] 0.053, ω → 45.6/λ}, CarrierFrequency → 45.6/λ, PointsPerCycle → n]]]}
   , {λ, 770, 830, 10}, {n, 100, 1000, 100}]
```

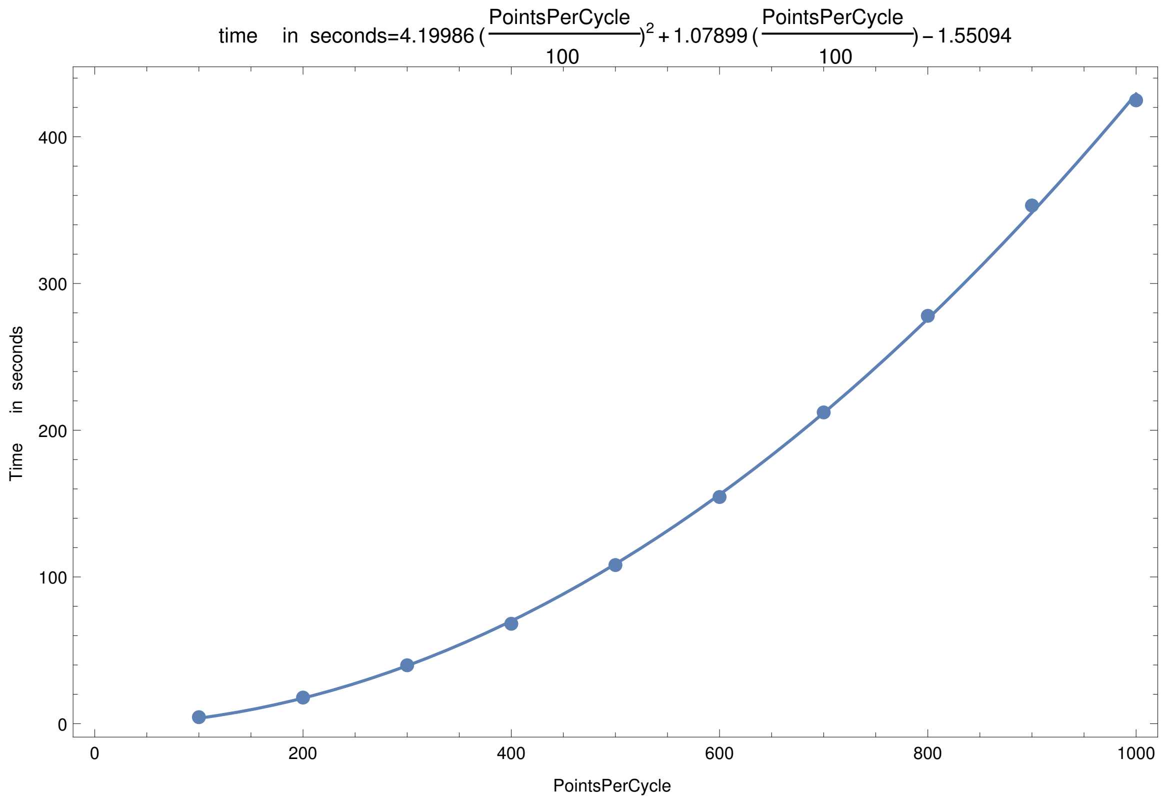
```
{{{100, {4.71833, 7949312}}, {200, {18.2567, 19028920}},
   {300, {39.6818, 43507248}}, {400, {68.0825, 75529984}}, {500, {108.27, 118772272}},
   {600, {154.871, 173234856}}, {700, {211.182, 232622488}}, {800, {275.072, 301126208}},
   {900, {350.061, 387140984}}, {1000, {424.31, 473885664}}},
 {{100, {3.7169, 7949264}}, {200, {17.4824, 19028912}}, {300, {40.1556, 43507248}},
   {400, {68.1348, 75529984}}, {500, {109.399, 118772272}},
   {600, {153.631, 173234976}}, {700, {211.956, 232622248}},
   {800, {279.29, 301126208}}, {900, {353.1, 387141104}}, {1000, {429.039, 473885664}}},
 {{100, {4.70831, 7949264}}, {200, {17.3451, 19028912}}, {300, {40.1826, 43507248}},
   {400, {68.9951, 75529984}}, {500, {109.87, 118772272}}, {600, {156.143, 173234976}},
   {700, {209.361, 232622488}}, {800, {279.29, 301126208}},
   {900, {350.784, 387141104}}, {1000, {418.914, 473885664}}},
 {{100, {4.27991, 7949264}}, {200, {18.6273, 19028912}}, {300, {40.4553, 43507248}},
   {400, {67.5833, 75529984}}, {500, {107.493, 118772272}}, {600, {151.738, 173234976}},
   {700, {211.565, 232622368}}, {800, {276.237, 301126208}},
   {900, {352.376, 387141104}}, {1000, {429.842, 473885664}}},
 {{100, {4.80072, 7949264}}, {200, {17.9797, 19028912}}, {300, {38.2944, 43507248}},
   {400, {68.7844, 75529984}}, {500, {105.406, 118772272}}, {600, {151.986, 173234976}},
   {700, {216.018, 232622488}}, {800, {279.116, 301126208}},
   {900, {361.033, 387141104}}, {1000, {424.589, 473885664}}},
 {{100, {4.24956, 7949264}}, {200, {17.1782, 19028912}}, {300, {40.1991, 43507248}},
   {400, {67.2693, 75529864}}, {500, {108.852, 118772032}},
   {600, {156.147, 173234736}}, {700, {210.96, 232622008}}, {800, {279.971, 301126208}},
   {900, {352.399, 387141104}}, {1000, {423.345, 473885664}}},
 {{100, {4.4809, 7949264}}, {200, {17.6272, 19028912}}, {300, {39.652, 43507248}},
   {400, {67.7538, 75529984}}, {500, {107.413, 118772272}}, {600, {156.833, 173234976}},
   {700, {213.902, 232622488}}, {800, {276.711, 301126208}},
   {900, {352.447, 387140984}}, {1000, {423.858, 473885544}}}}
```

```
parallelTimingsListAveraged =
  Table[{parallelTimingsListᵀ[[k, 1, 1]], Mean[parallelTimingsListᵀ[[k, All, 2]]]},
   {k, Length[parallelTimingsListᵀ]}]
```

```
{{100, {4.42209, 55644896/7}}, {200, {17.7852, 133202392/7}},
 {300, {39.803, 43507248}}, {400, {68.0862, 528709768/7}}, {500, {108.101, 831405664/7}},
 {600, {154.479, 1212644472/7}}, {700, {212.135, 232622368}}, {800, {277.955, 301126208}},
 {900, {353.171, 2709987488/7}}, {1000, {424.842, 3317199528/7}}}
```

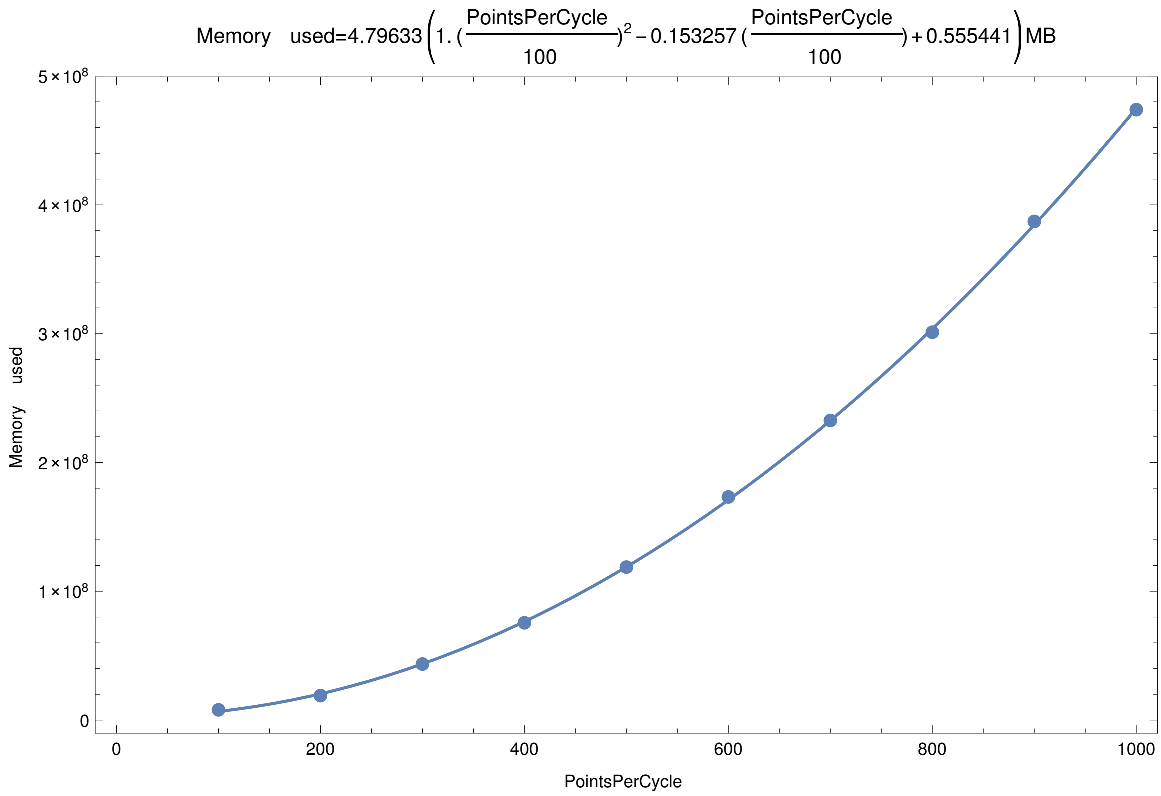
Timings

```
parallelTimingsModel =
    LinearModelFit[(Flatten/@parallelTimingsListAveraged)[[All, {1, 2}]], {1, n, n^2}, {n}];
Show[{
    ListPlot[
      (Flatten/@parallelTimingsListAveraged)[[All, {1, 2}]]
    ],
    Plot[parallelTimingsModel [n],
      {n, parallelTimingsListAveraged[[1, 1]], parallelTimingsListAveraged[[-1, 1]]}]
  }
  , Frame → True,
  PlotLabel → Row[{"time  in seconds=", parallelTimingsModel [100 " (PointsPerCycle/100) "]}],
  FrameLabel → {"PointsPerCycle", "Time  in seconds"}, ImageSize → 600
]
```



Memory

```
parallelMemoryModel =
    LinearModelFit[(Flatten/@parallelTimingsListAveraged)[[All, {1, 3}]], {1, n, n^2}, {n}];
Show[{
    ListPlot[
      (Flatten/@parallelTimingsListAveraged)[[All, {1, 3}]]
    ],
    Plot[parallelMemoryModel [n],
      {n, parallelTimingsListAveraged[[1, 1]], parallelTimingsListAveraged[[-1, 1]]}]
  }
  , Frame →True, PlotLabel→
    Row[{"Memory  used=", Simplify [parallelMemoryModel [100. "(PointsPerCycle/100)"] 10^-6 "MB"]}],
  FrameLabel →{"PointsPerCycle", "Memory  used"}, ImageSize →600
]
```



Decoupling integration and sampling

If the quadratic scaling with respect to PointsPerCycle becomes too onerous, the sampling rate for the evaluation and for the numerical integration can be decoupled by providing an explicit option for IntegrationPointsPerCycle, which will set how many points are used per cycle in the numerical integration.

```
?IntegrationPointsPerCycle
```

IntegrationPointsPerCycle is an option for makeDipoleList which controls the number of points per cycle to use for the integration. Set to Automatic, to follow PointsPerCycle, or to an integer.

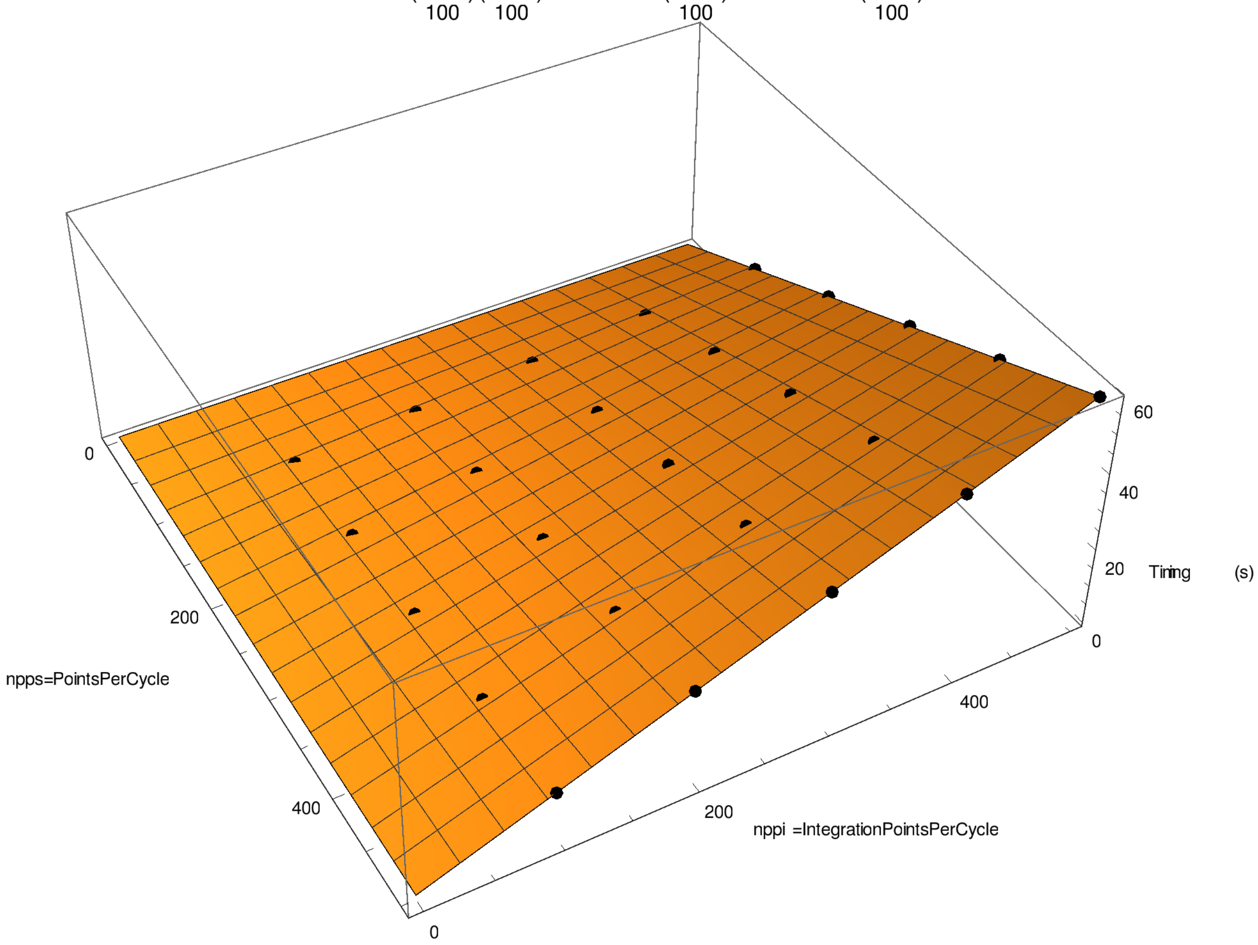
Obviously, if this option is taken, then the outcome should be checked for numerical convergence with respect to IntegrationPointsPerCycle.

```
DateString[]
decoupledTimingsList = Table[
  {nSampling , nIntegration, AbsoluteTiming [
     MaxMemoryUsed [makeDipoleList [VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}]
        , FieldParameters →{F→ Sqrt[nSampling/100] 0.053, ω→0.057}
        , PointsPerCycle→nSampling , IntegrationPointsPerCycle→nIntegration
      ]]]}
  , {nSampling , 100, 500, 100}, {nIntegration, 100, 500, 100}]
DateString[]
```

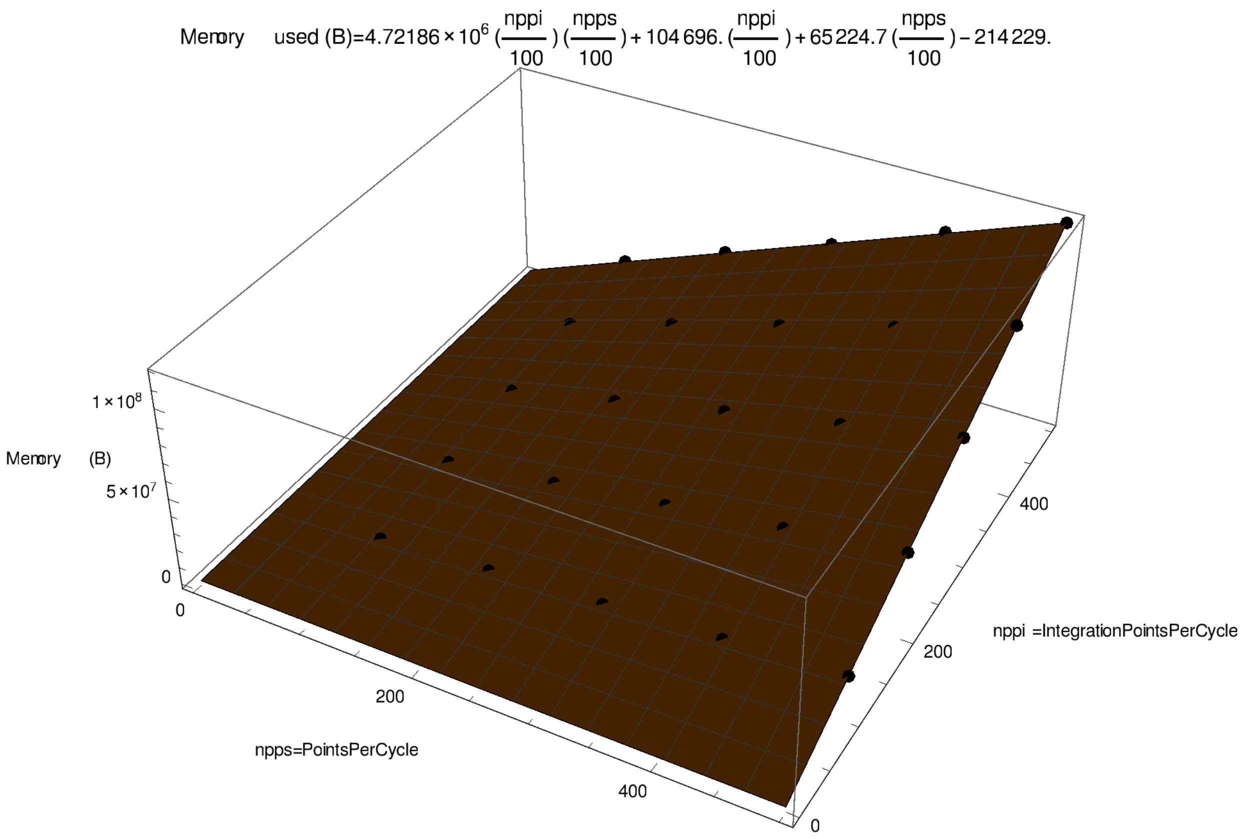
```
Fri 5 Feb 2016 17:08:36

{{{100, 100, {2.39139, 4789112}}, {100, 200, {4.76862, 9473168}}, {100, 300, {7.07336, 14241584}},
    {100, 400, {9.39992, 19138656}}, {100, 500, {11.7964, 24035936}}},
  {{200, 100, {4.72937, 9447040}}, {200, 200, {9.3758, 19017072}},
    {200, 300, {14.1647, 28478336}}, {200, 400, {18.8825, 38195832}},
    {200, 500, {23.4484, 47916856}}}, {{300, 100, {7.02226, 14177120}},
    {300, 200, {14.067, 28434560}}, {300, 300, {21.5271, 43233320}},
    {300, 400, {28.4153, 56993480}}, {300, 500, {35.2399, 70747672}}},
  {{400, 100, {9.41508, 19027632}}, {400, 200, {18.6957, 38108104}},
    {400, 300, {28.159, 56949288}}, {400, 400, {37.5268, 75259432}},
    {400, 500, {46.83, 95676600}}}, {{500, 100, {11.6672, 23882240}},
    {500, 200, {23.5225, 47786456}}, {500, 300, {35.0966, 70660928}},
    {500, 400, {46.7308, 95634048}}, {500, 500, {58.5078, 118501720}}}}

Fri 5 Feb 2016 17:17:24
```

```
decoupledTimingsModel = LinearModelFit[
    (Flatten /@Flatten[decoupledTimingsList, {{1, 2}}]) [[All, {1, 2, 3}]]
    , {1, nSampling , nIntegration, nSampling ×nIntegration}, {nSampling , nIntegration}];
Show[{
   Plot3D[
     decoupledTimingsModel [nSampling , nIntegration]
     , {nSampling , 0, 500}, {nIntegration, 0, 500}
     , AxesLabel → {"npps=PointsPerCycle", "nppi=IntegrationPointsPerCycle", "Timing  (s)"}
     , PlotLabel →
       Row[{"time  in seconds=", decoupledTimingsModel [100 " (npps/100) ", 100 " (nppi/100) "]}]
     , ImageSize → 650
   ],
   ListPointPlot3D[
     (Flatten /@Flatten[decoupledTimingsList, {{1, 2}}]) [[All, {1, 2, 3}]]
     , PlotStyle → {{Black, PointSize[Large]}}
   ]
 }]
```

time in seconds=$2.33613 \left(\frac{\text{nppi}}{100}\right)\left(\frac{\text{npps}}{100}\right)+0.0260245\left(\frac{\text{nppi}}{100}\right)-0.00404974\left(\frac{\text{npps}}{100}\right)+0.0470921$



```
decoupledMemoryModel = LinearModelFit[
    (Flatten /@Flatten[decoupledTimingsList, {{1, 2}}]) [[All, {1, 2, 4}]]
    , {1, nSampling , nIntegration, nSampling ×nIntegration}, {nSampling , nIntegration}];
Show[{
   Plot3D[
     decoupledMemoryModel [nSampling , nIntegration]
     , {nSampling , 0, 500}, {nIntegration, 0, 500}
     , AxesLabel → {"npps=PointsPerCycle", "nppi=IntegrationPointsPerCycle", "Memory  (B)"}
     , PlotLabel →
       Row[{"Memory  used (B)=", decoupledMemoryModel [100 "(npps/100)", 100 "(nppi/100)"]}]
     , ImageSize → 650
   ],
   ListPointPlot3D[
     (Flatten /@Flatten[decoupledTimingsList, {{1, 2}}]) [[All, {1, 2, 4}]]
     , PlotStyle → {{Black, PointSize[Large]}}
   ]
 }]
```



Cutting off the long trajectories

This section shows, as an example of the use of the package, the use of the integration gate to eliminate the contribution from long trajectories. This can be tested by the reduced presence of quantum path interference patterns in the spectrum, and more practically by examining the dependence of the quantum phase on the field intensity.

The gating cutoff time

Given the classical trajectory,

```
trajectory[ωt_, ωt0_] := (x[ωt] /. First@DSolve[
            {x''[ωt] == Cos[ωt], x'[ωt0] == 0, x[ωt0] == 0}
            , x, ωt
          ])
```

the recollision kinetic energy and excursion time can be found as

```
recollisionKE[ωt0_?NumericQ ] := (D[trajectory[ωtt, ωt0], ωtt]^2 /. ωtt → ωt) /.
     First[Quiet[NSolve[{trajectory[ωt, ωt0] == 0, π/2 ≤ ωt < 2π}, ωt]]]
recollisionExcursionTime[ωt0_?NumericQ ] :=
   (ωt - ωt0) /. First[Quiet[NSolve[{trajectory[ωt, ωt0] == 0, π/2 ≤ ωt < 2π}, ωt]]]
```

and the excursion time at the cutoff can be found by maximizing the kinetic energy.

```
FindMaximum [recollisionKE[ωt0], {ωt0, 0.3}]
recollisionExcursionTime[ωt0]/(2π) /. Last[%]
```

```
{1.58657, {ωt0 → 0.313408}}

0.650239
```

In other words, the cutoff trajectories occur at excursion times of $\omega\tau = 0.65\times 2\pi$, i.e. at a gate number of 0.65 cycles.

Calculation

This calculation runs a standard linearly-polarized field with intensity between 0.8 and $2\times 10^{14}\ \mathrm{W/cm}^2$, with a fine intensity resolution. We compare the standard, non-gated calculation against a calculation with nGate set to 0.65, as per the above, and a sharp $\sin^2$ cutoff of 0.05 cycles.

```
intRange = Range[0.8, 2., 0.002];
nppsl = 150;
SetSharedFunction[quantumPhaseScan , fourierDipole];
LaunchKernels[];
nFlat = 0.65;
nGateRamp  = 0.05;
```

```
LaunchKernels::nodef : Some subkernels are already running. Not launching default kernels again. >>
```

The actual calculation,

```
DateString[]
AbsoluteTiming [
  ParallelTable[
      quantumPhaseScan [trajectories, int] = makeDipoleList [
            VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}],
            FieldParameters →{F→√int 0.053, ω→0.057},
            PointsPerCycle→nppsl,
            If[trajectories=== "Short", Sequence@@{Gate→SineSquaredGate[nGateRamp ],
                  nGate→ (nFlat+nGateRamp )}, ##&[], ##&[]]
          ];
      , {int, intRange}, {trajectories, {"Short", "Long"}}];
]
DateString[]
```

```
Wed 2 Dec 2015 19:28:50

{1216.8, Null}

Wed 2 Dec 2015 19:49:07
```

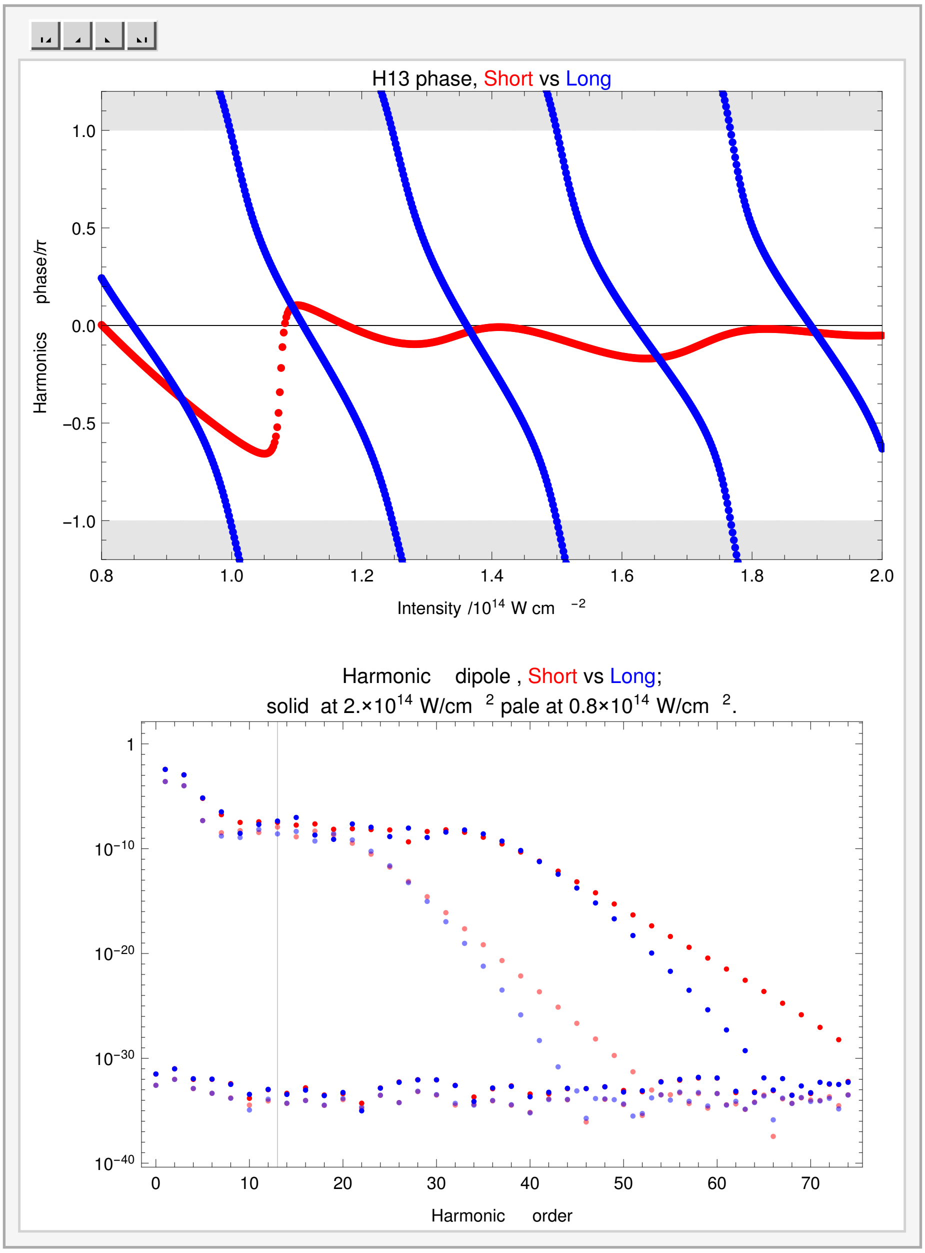
and the energy-domain dipole.

```
AbsoluteTiming [
  Table[
      fourierDipole[trajectories, int] = Fourier[
            Re[quantumPhaseScan [trajectories, int][[1 ;; -2, 1]]]
            , FourierParameters →{-1, 1}];
      , {int, intRange}, {trajectories, {"Short", "Long"}}];
]
```

```
{0.093762, Null}
```

Analysis

```
Block[{background},
  background = ListLogPlot[
      Flatten[Table[
          {Range[0, nppsl/2 - 1], Abs[fourierDipole[trajectories, m [intRange]][[1 ;; nppsl/2]]]^2}^T
          , {m , {Min, Max}}, {trajectories, {"Short", "Long"}}], 1]
      , ImageSize → 420
      , PlotStyle → {{Red, Opacity[0.5]}, {Blue, Opacity[0.5]}, {Red}, {Blue}}
      , PlotLabel → "Harmonic  dipole, Short vs Long;\nsolid at " <> ToString[Max[intRange]] <>
          "×10^14  W/cm ^2  pale at " <> ToString[Min[intRange]] <> "×10^14  W/cm ^2."
      , FrameLabel  → {"Harmonic  order", ""}
      , Frame  → True
    ];
  SlideView[Table[
      Row[{
          Show[{
              RegionPlot[Abs[ϕ] > 1, {int, Min[intRange], Max[intRange]}, {ϕ, -1.2, 1.2}, PlotStyle
                  GrayLevel[0.9], Method → {"AxesInFront" → False}, BoundaryStyle → None],
              ListPlot[
                Table[
                  Flatten[Table[
                      {#, {#[[1]], #[[2]] + 2}, {#[[1]], #[[2]] - 2}} &@
                        {int, 1/π Arg[fourierDipole[trajectories, int][[HO + 1]]]}
                      , {int, intRange[[1 ;; -1]]}], 1]
                  , {trajectories, {"Short", "Long"}}]
                , PlotStyle → {{PointSize[0.01], Red}, {PointSize[0.01], Blue}}
                , Joined → False
              ]
            }
            , PlotRange → 1.2 {-1, 1}, AspectRatio → 0.6
            , PlotRangePadding → {None, Automatic }
            , Axes → True
            , ImageSize  → 450
            , PlotLabel → "H" <> ToString[HO] <> " phase,  Short vs Long"
            , FrameLabel  → {"Intensity/10^14  W cm ^-2", "Harmonics  phase/π"}
          ],
          Show[{background}, GridLines → {{HO}, None}]
        }]
      , {HO, 1, nppsl/2, 2}], 7]
]
```



The short- and long-trajectory calculations are in red and blue respectively. It is clear that, in the plateau regions, the gated calculation has a much smoother dependence of the harmonic phase on the field intensity. On the other hand, the cutoff is perfectly preserved. These are the hallmarks that the contributions from long trajectories have been mostly eliminated.

Nondipole contributions

Nondipole contributions can be specified by setting a nonzero vector potential gradient:

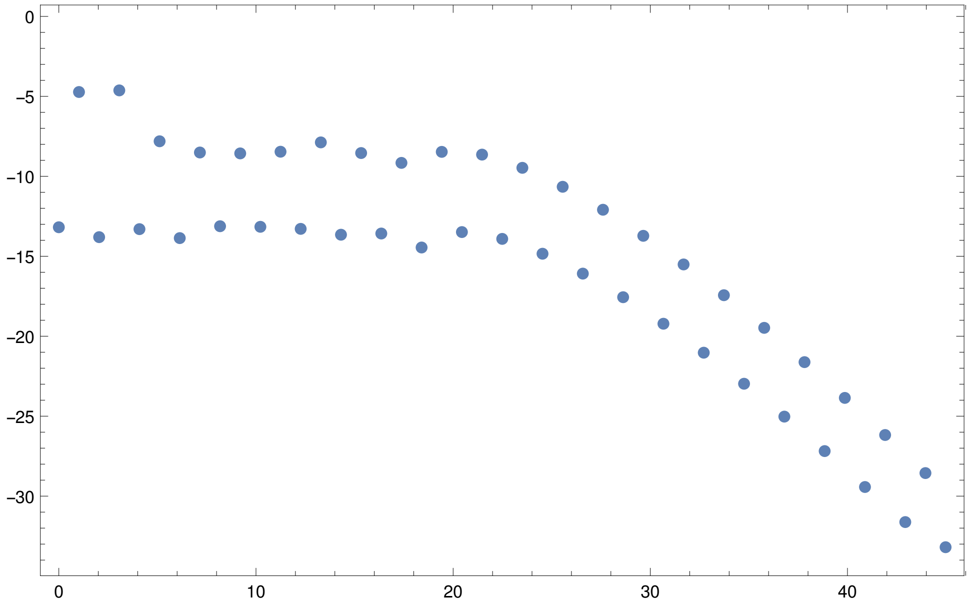
```
?VectorPotentialGradient
```

"VectorPotentialGradient is an option for makeDipole list which specifies the gradient of the field's vector potential. Usage should be VectorPotentialGradient →GA, where GA[t]//.pars must yield a square matrix of the same dimension as the vector potential for numeric t and parameters indicated by FieldParameters →pars. The indices must be such that GA[t]⟦i ,j⟧ returns $\partial_i A_j[t]$."

If, for example, the travelling-wave form of the vector potential is of the form $\boldsymbol{A}(\boldsymbol{r},\ t) = \frac{F}{\omega}\,\hat{\boldsymbol{x}}\cos(k\,z - \omega\,t)$, then at the origin the vector potential is $\boldsymbol{A}(\mathbf{0},\ t) = \frac{F}{\omega}\hat{\boldsymbol{x}}\cos(\omega\, t)$ and it has a single nonzero entry in its gradient matrix $\nabla \boldsymbol{A}$, i.e. $\partial_z A_x = -\frac{k F}{\omega}\sin(\omega t)$. This is entered into the VectorPotentialGradient option as

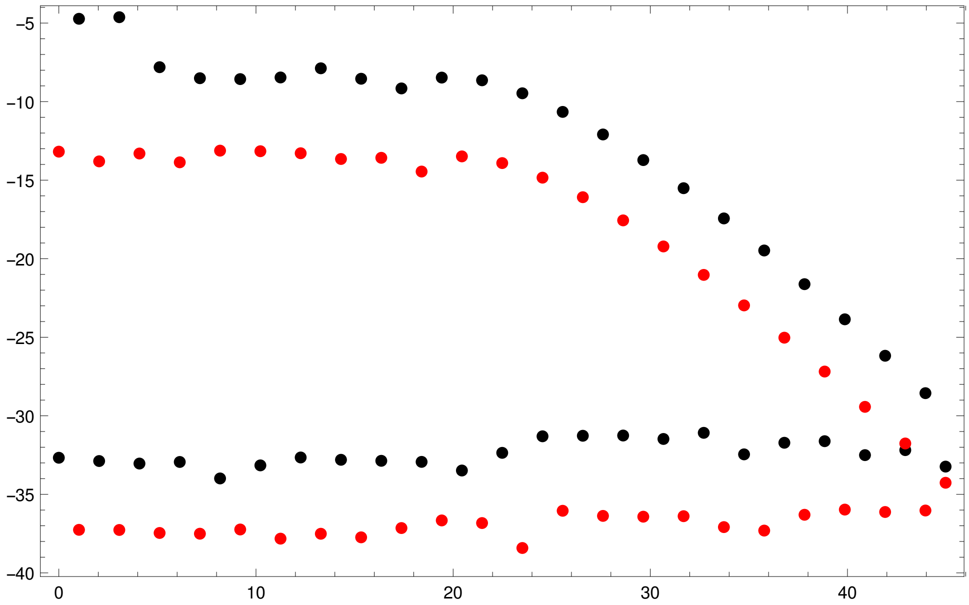
```
AbsoluteTiming[
  nonDipoleContributions = makeDipoleList[
        VectorPotential → Function[t, {F/ω Cos[ω t], 0, 0}],
        VectorPotentialGradient → Function[t, {{0, 0, 0}, {0, 0, 0}, {-k F/ω Sin[ω t], 0, 0}}],
        FieldParameters → {F → 0.05, ω → 0.057, k → ω/c, c → 137}
      ];
]
{4.8337, Null}
```

```
spectrumPlotter[getSpectrum[Most[nonDipoleContributions]], Joined → False, ImageSize → 500]
```



At low wavelengths, the first obvious effect is the appearance of even harmonics. This is the expected behaviour, with the harmonics along the laser propagation direction. (Informally, the magnetic pushing on the wavepacket acts on the propagation direction on both halves of each laser period. This off-axis recollision causes the dipole to oscillate in the propagation direction with an even symmetry. More formally, the dynamical symmetries of the problem permit even (but not odd) harmonics along this direction.) This is indeed what is observed:

```
Show[{
    spectrumPlotter [getSpectrum [nonDipoleContributions⟦1 ;; -2, {1, 2}⟧],
      Joined→ False, PlotStyle→Black],
    spectrumPlotter [getSpectrum [nonDipoleContributions⟦1 ;; -2, {3}⟧],
      Joined→ False, PlotStyle→Red]
  }, PlotRange→All, ImageSize →500]
```



Benchmarking the nondipole contributions

Nondipole contributions in a crossed-beam setup

This section explores the harmonic emission by a crossed-beam setup, with nondipole contributions, as a benchmarking step for the latter. The crossed-beam setup was proposed by X.-M. Tong and S.-I. Chu in *Phys. Rev. A* **58** no .4, R2656 (1998), and it was explored in a nondipole setting by V. Averbukh et al. in *Phys Rev. A* **65,** 063402 (2002). The results below reproduce those of Averbukh et al.

In short, we consider the harmonic emission by a circularly polarized pulse propagating along the z direction, at frequency ω, and a linearly polarized pulse of frequency $r\,\omega$ propagating along the x direction and polarized along the z direction.

```
(crossedBeamsA [x_, z_] = Function[t,
      {F1/ω Cos[k z - ω t], F1/ω Sin[k z - ω t], F2/(r ω) Sin[r k x - r ω t + θ0]}
    ]) [t] // MatrixForm
(crossedBeamsGA [x_] = Function[t, Evaluate[{
          D[crossedBeamsA [x, z][t], x] /. {z → 0},
          D[crossedBeamsA [x, z][t], y] /. {z → 0},
          D[crossedBeamsA [x, z][t], z] /. {z → 0}
        }]]
  ) [t] // MatrixForm
```

$$\begin{pmatrix} \frac{F1\,Cos[k\,z-t\,\omega]}{\omega} \\ \frac{F1\,Sin[k\,z-t\,\omega]}{\omega} \\ \frac{F2\,Sin[k\,r\,x+\theta0-r\,t\,\omega]}{r\,\omega} \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & \frac{F2\,k\,Cos[k\,r\,x+\theta0-r\,t\,\omega]}{\omega} \\ 0 & 0 & 0 \\ \frac{F1\,k\,Sin[t\,\omega]}{\omega} & \frac{F1\,k\,Cos[t\,\omega]}{\omega} & 0 \end{pmatrix}$$

The dipole selection rules allow harmonic orders of the form $2\,r\,l \pm 1$, with $l = 0, 1, 2, 3, ...$, with polarization in the x, y plane, and harmonics of order $r(2\,l+1)$, with $l = 0, 1, 2, 3, ...$, polarized along the z direction.

```
allowedHarmonics [r_, {1, 2}] := Select[Union[2 r Range[0, 100] + 1, 2 r Range[0, 100] - 1], # > 0 &]
allowedHarmonics [r_, {3}] := r (2 Range[0, 100] + 1)
```

For the calculation, then, some preliminaries,

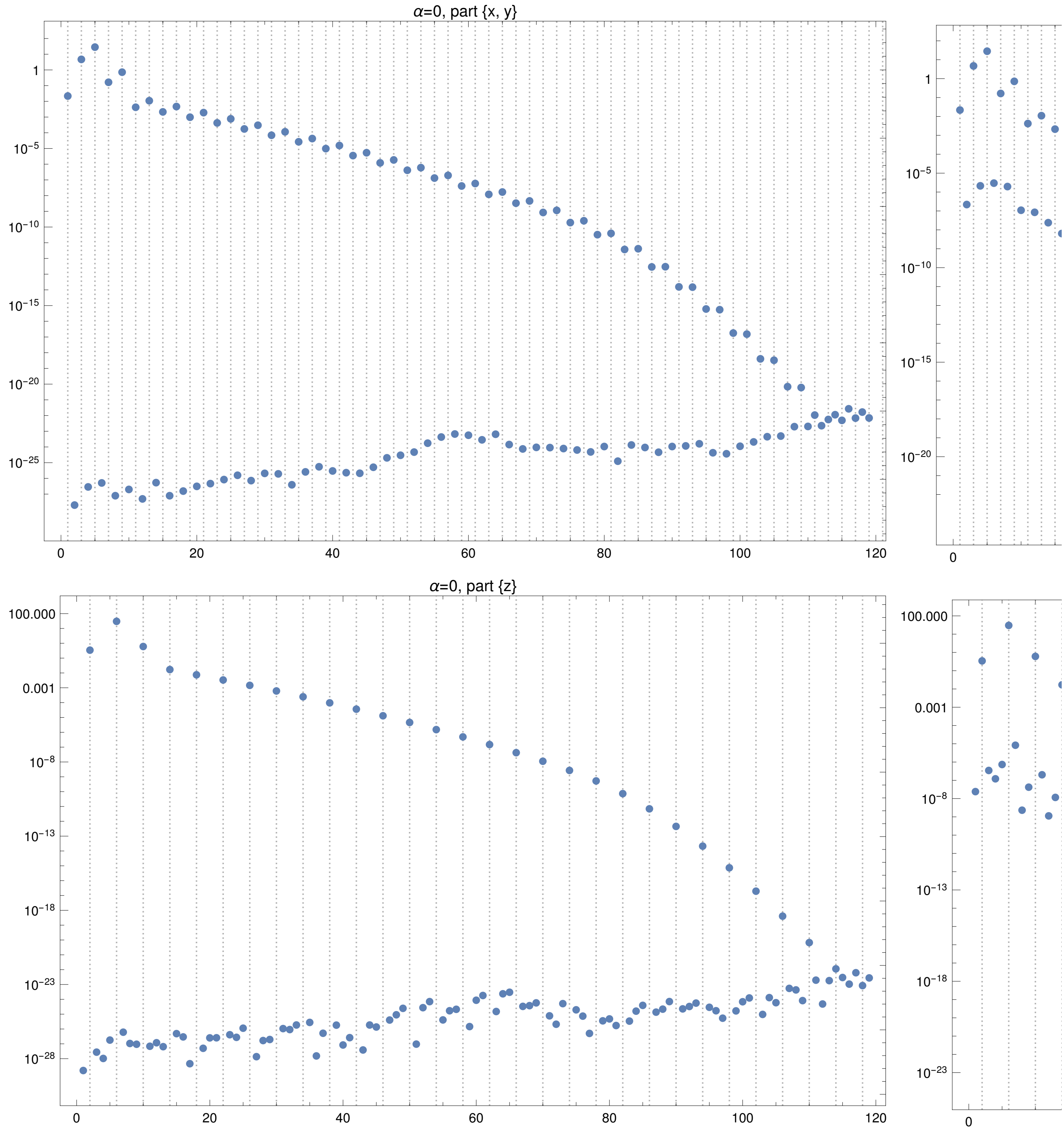
```
αRange = {0, 1 / 137};
nppcb = 240;
crossedBeamsParameters [rr_] := {F1 → 0.1, F2 → 0.2, ω → 0.057, θ0 → 0, r → rr, k → α ω};
SetSharedFunction[crossedBeamsResults ];
```

and the calculation itself for $r = 2$ and $r = 5$.

```
Print[DateString[]]
ParallelTable[AbsoluteTiming [
    crossedBeamsResults [r, α] = makeDipoleList [
         VectorPotential → crossedBeamsA [0, 0], VectorPotentialGradient → crossedBeamsGA [0],
         FieldParameters → crossedBeamsParameters [r]
         , DipoleTransitionMatrixElement → Function[{p, κ}, gaussianDTME[p, 1 / 1.3]]
         , CarrierFrequency → 0.057, PointsPerCycle → nppcb
       ];
  ], {r, {2, 5}}, {α, αRange}]
Print[DateString[]]

Fri 13 May 2016 17:24:01

{{{23.0971, Null}, {52.5426, Null}}, {{22.9569, Null}, {52.8564, Null}}}

Fri 13 May 2016 17:25:19
```

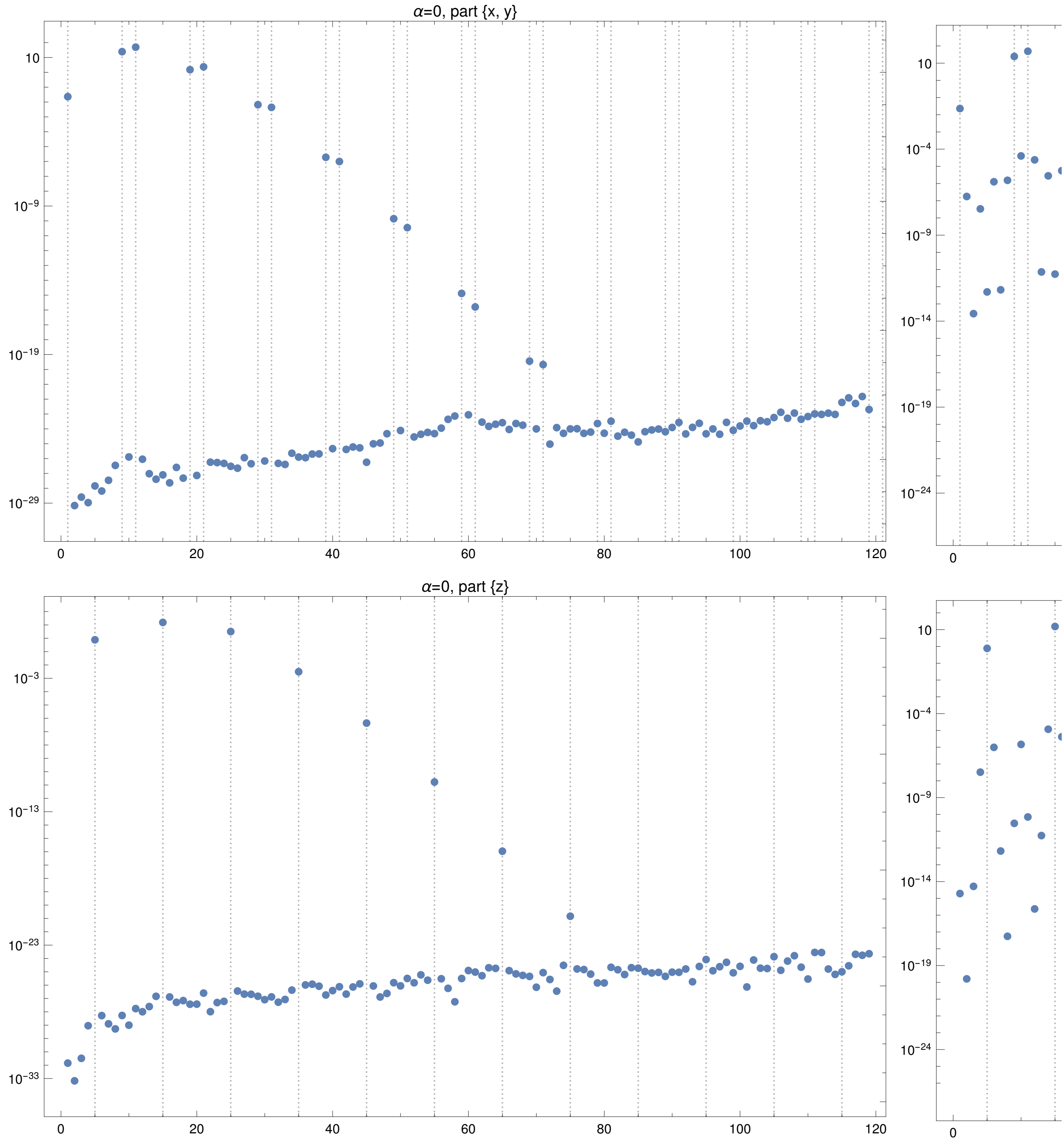
Results for $r = 2$, comparable to Fig. 1 in Averbukh et al. Dashed lines mark the dipole-allowed harmonics. The left-hand column has nondipole contributions turned off ($\alpha = 0$), and the right-hand column includes the nondipole contributions and observes a massive increase in the amplitude of the dipole-forbidden harmonics.

```
Grid[Table[
   ListLogPlot[
    getSpectrum [crossedBeamsResults [2, α]〚1 ;; -2, part〛, ωPower → 2]〚2 ;;〛
    , Joined → False, ImageSize → 600, PlotTheme → "Detailed", PlotRange → Full
    , GridLines → {allowedHarmonics [2, part], None}
    , PlotLabel → Row[{"α=", α, ", part ", {"x", "y", "z"}〚part〛}]
   ], {part, {{1, 2}, {3}}}, {α, αRange}]]
```



Results for $r = 5$, comparable to Fig. 2 in Averbukh et al.

```
Grid[Table[
   ListLogPlot[
    getSpectrum [crossedBeamsResults [5, α]⟦1 ;; -2, part⟧, ωPower → 2]⟦2 ;;⟧
    , Joined→ False, ImageSize → 600, PlotTheme → "Detailed", PlotRange → Full
    , GridLines→ {allowedHarmonics [5, part], None}
    , PlotLabel → Row[{"α=", α, ", part ", {"x", "y", "z"}⟦part⟧}]
  ], {part, {{1, 2}, {3}}}, {α, αRange}]]
```



Multiple plateaus in HHG in ions

This section benchmarks this code against the results of N.J. Kylstra et al. reported in *J. Phys B: At. Mol. Opt. Phys.* **34** no. 3, L55 (2001);. In particular, we study HHG in the He^+ ion at high intensity ($I = 5.6\times10^{15}\,\mathrm{W\,cm}^{-2}$) and reasonable (800nm) wavelength.

```
(kylstraA[z_] = Function[t, {F/ω Sin[(ω t - k z)/4]^2 Sin[ω t - k z], 0, 0}])[t] // MatrixForm
(kylstraGA = Function[t, Evaluate[{
              D[kylstraA[z][t], x] /. {z → 0},
              D[kylstraA[z][t], y] /. {z → 0},
              D[kylstraA[z][t], z] /. {z → 0}
            }]]
    )[t] // MatrixForm
```

$$\begin{pmatrix} -\frac{F\,\mathrm{Sin}[k z - t\omega]\,\mathrm{Sin}\left[\frac{1}{4}(-k z + t\omega)\right]^2}{\omega} \\ 0 \\ 0 \end{pmatrix}$$

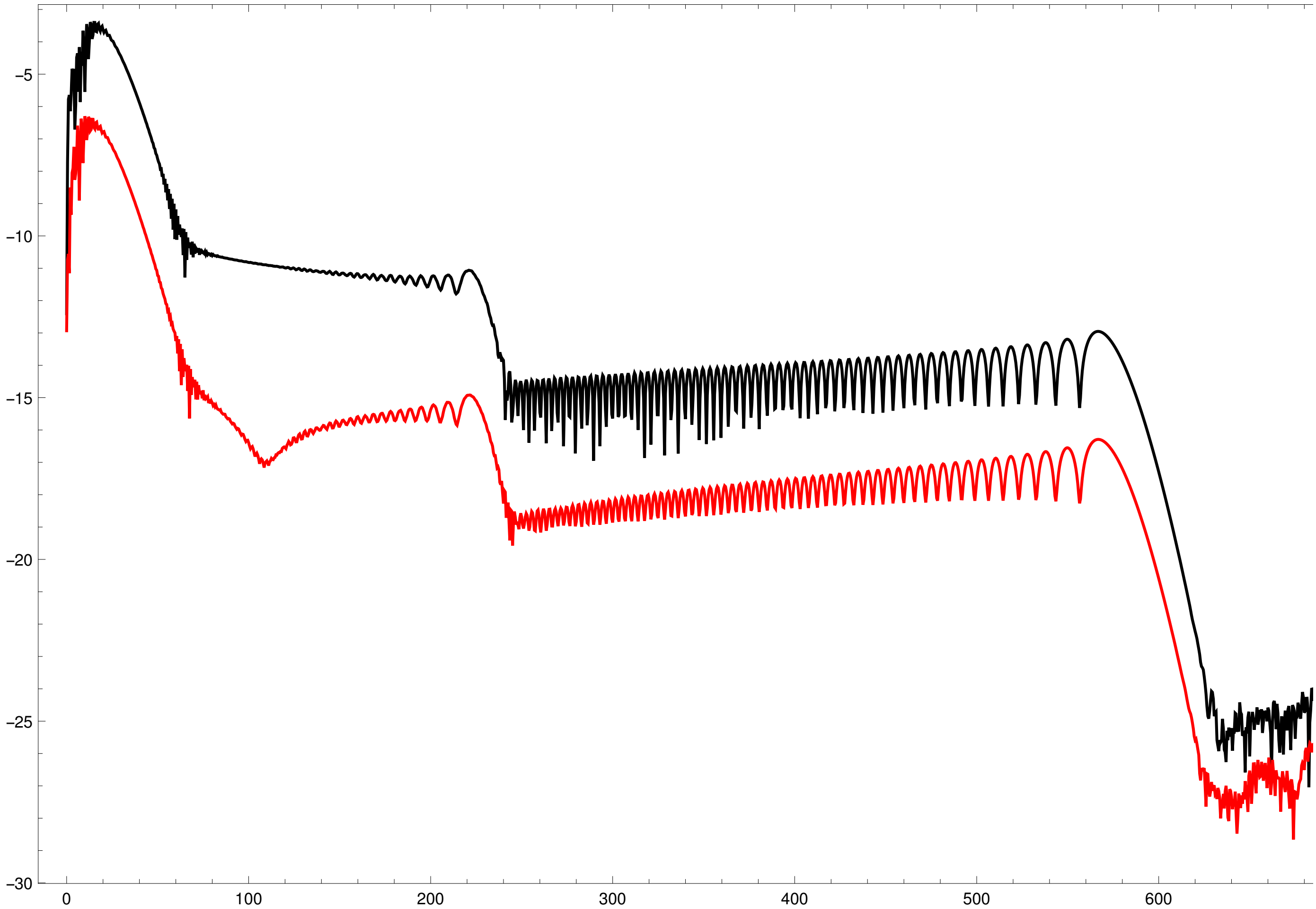
$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -\frac{F k\,\mathrm{Cos}[t\omega]\,\mathrm{Sin}\left[\frac{t\omega}{4}\right]^2}{\omega} - \frac{F k\,\mathrm{Cos}\left[\frac{t\omega}{4}\right]\mathrm{Sin}\left[\frac{t\omega}{4}\right]\mathrm{Sin}[t\omega]}{2\omega} & 0 & 0 \end{pmatrix}$$

```
nppk = 1500;

DateString[]
AbsoluteTiming[
  kylstraTest = makeDipoleList[
        VectorPotential→kylstraA[0], VectorPotentialGradient→kylstraGA,
        FieldParameters →{F→Sqrt[5.6×10^15/10^14] 0.053, ω→0.057, k→αω, α→1/137},
        IonizationPotential→2,
        PointsPerCycle→nppk, TotalCycles→2
      ];
]
DateString[]
```

```
Thu 1 Oct 2015 18:25:47

{1619.16, Null}

Thu 1 Oct 2015 18:52:46
```

Plotting the results. The *x* component (along the laser polarization) is in black, the *z* component (along the laser propagation) is in red.

```
Show[
  Table[
    spectrumPlotter [getSpectrum [kylstraTest⟦1 ;; -2, part⟧, ωPower→2], Joined→True,
      PointsPerCycle→nppk, TotalCycles→2, PlotStyle→ (part /. {{1, 2}→Black, {3}→Red})]
    , {part, {{1}, {3}}}]
  , PlotRange→All
]
```



The results are a good qualitative match to the dipoles reported by Kylstra et al., with the notable exception of the low-order harmonics below n≲50.

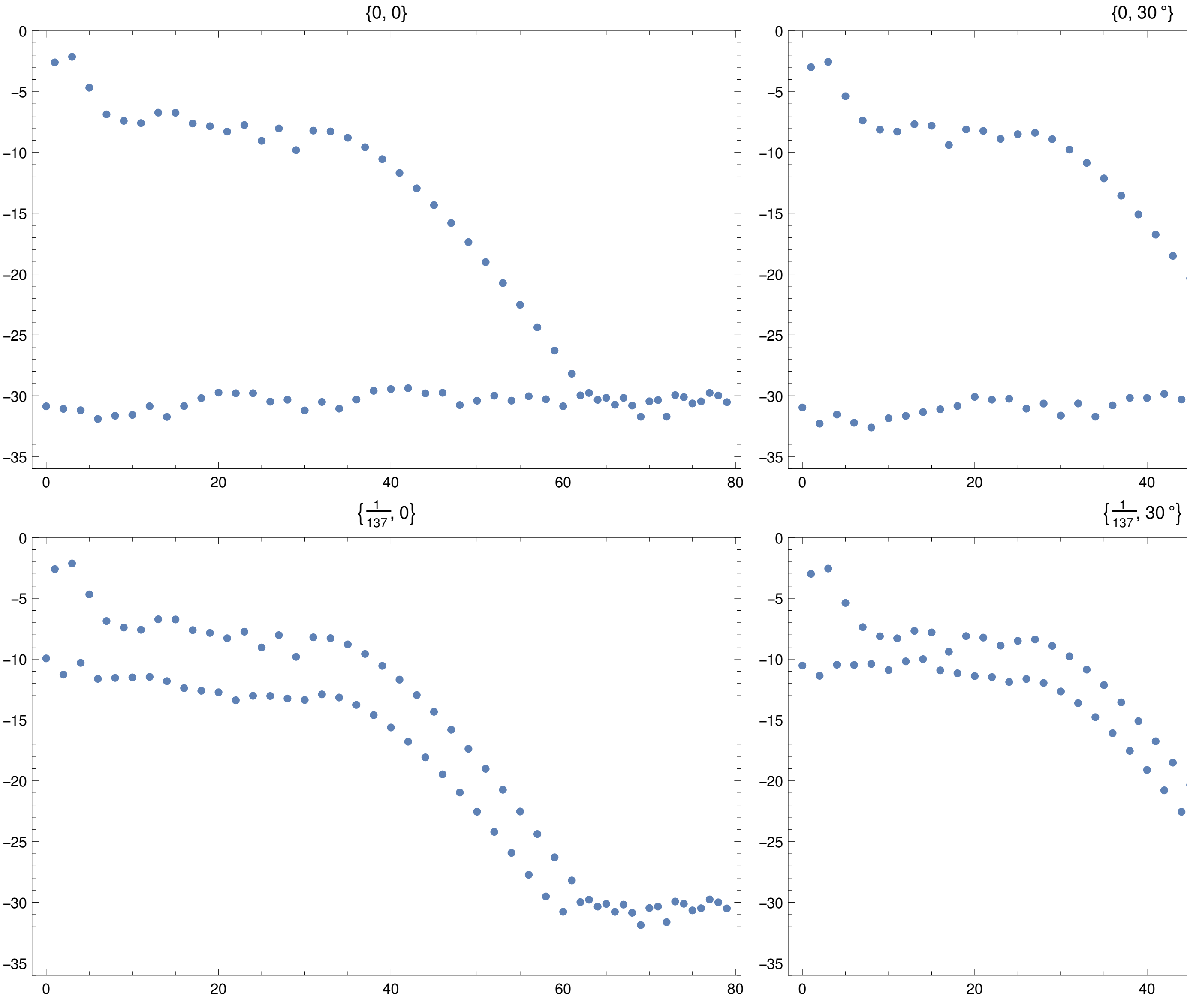
On the other hand, taken naively this code cannot be applied to the harder targets described in that paper (Li^{2+} and Be^{3+}, at intensities between 0.9 and $3.6\times10^{17}\,\mathrm{W\,cm^{-2}}$), which have cutoffs of order as high as 35 000, which requires several days to several months of calculation using the naive scaling. (That said, using a smarter choice of ReportingFunction, judicious use of parallelization and lots of waiting, those targets are probably within reach of this code.)

Periodicity of nondipole contributions

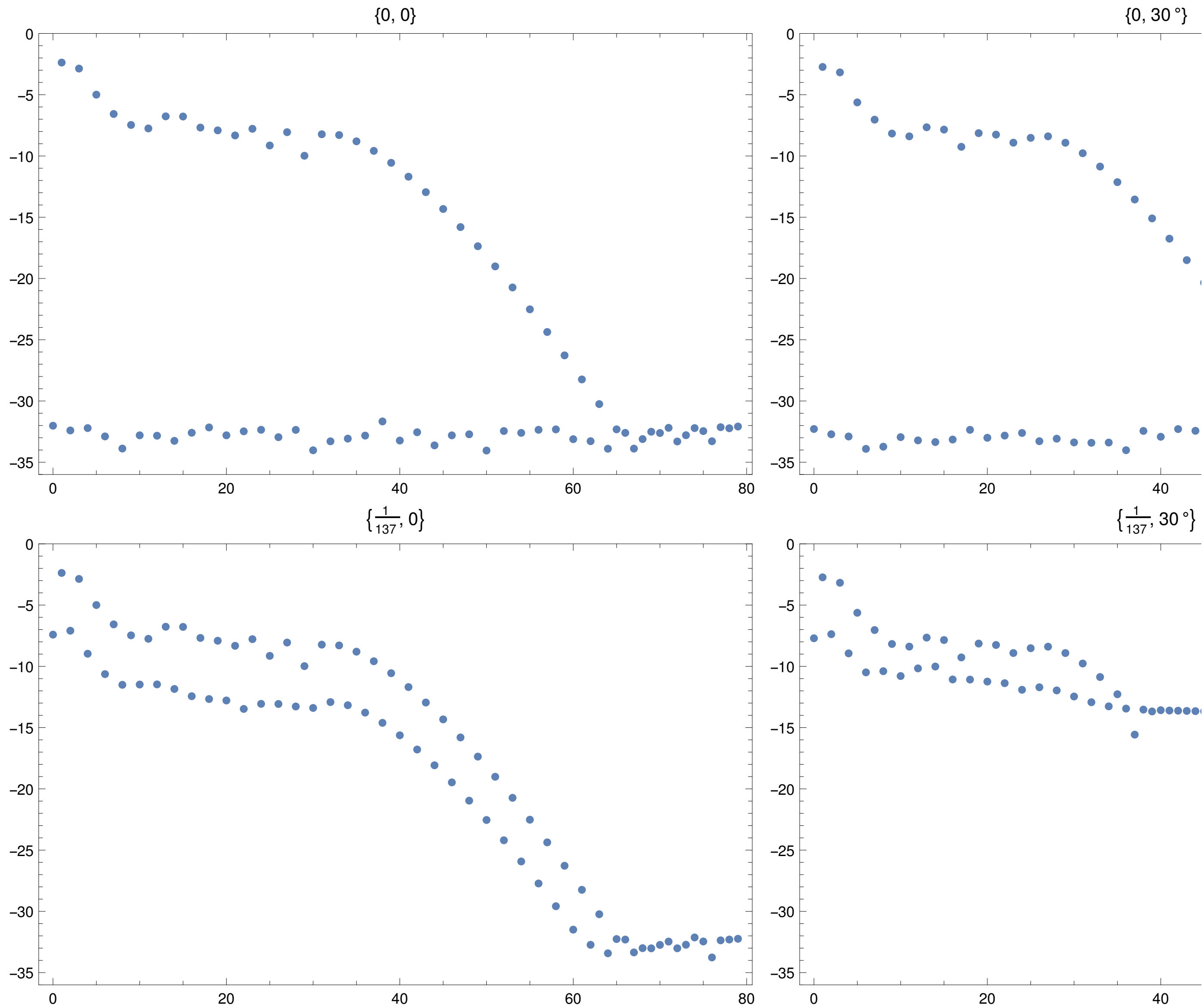
```
Quit
```

```
(crossedBeamsA [t_] = First/@Sum [
        F/ω {{Cos[θ]}, {0}, {-s Sin[θ]}} Cos[k {s Sin[θ], 0, Cos[θ]}.{x, y, z} - ω t - s ϕ]
        , {s, {-1, 1}}]) // MatrixForm ;
(crossedBeamsGA [t_] = Evaluate[{
        D[crossedBeamsA [t], x],
        D[crossedBeamsA [t], y],
        D[crossedBeamsA [t], z]
      }]) // MatrixForm ;
crossedBeamsParameters = {x → 0, y → 0, z → 0, F → Sqrt[0.5] 0.053, ω → 0.057, θ → 10. °, k → α ω};

DateString[]
Table[
  First[AbsoluteTiming [
      symmetryTestDipole [α, ϕ] = makeDipoleList [
          VectorPotential→ crossedBeamsA , VectorPotentialGradient→ crossedBeamsGA ,
          FieldParameters → crossedBeamsParameters ,
          PointsPerCycle→ 160
        ];
    ]]
  , {α, {0, 1/137}}, {ϕ, {0, 30 °}}]
DateString[]
Mon 2 May 2016 16:23:09

{{8.75144, 9.80629}, {11.9277, 16.5271}}

Mon 2 May 2016 16:23:56
```

```
Grid[Table[
    spectrumPlotter [
      getSpectrum [symmetryTestDipole  [α, ϕ][[1 ;; -2, {1, 2, 3}]]]
      , Joined→ False, PointsPerCycle→ 160,
      ImageSize  → 450, PlotLabel → {α, ϕ}, PlotRange → {-36, 0}
    ]
    , {α, {0, 1/137}}, {ϕ, {0, 30 °}}]]
```



Compare this with the unacceptably high noise floor from the previous version for the case $\alpha = 1/137$, $\phi = 30\,°$.



Debugging and benchmarking tools

If something goes funny with your calls, then before you start taking makeDipoleList apart you can try using its Verbose option to diagnose the internal functions it is using. In particular:

- Setting Verbose→1 makes makeDipoleList print the Information of the key internal functions it is using, before it goes on to the integration loop.

```
makeDipoleList[VectorPotential→Function[t, {F/ω Sin[ω t], 0, 0}],
    FieldParameters →{F→0.05, ω→0.057}, Verbose→1][[1 ;; 10]]
```

```
RBSFA`Private`A
```

```
RBSFA`Private`A[RBSFA`Private`t$_] = {0.877193 Sin[0.057 RBSFA`Private`t$], 0, 0}
```

```
RBSFA`Private`GA
```

```
RBSFA`Private`GA[RBSFA`Private`t$_] = {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}
```

```
RBSFA`Private`ps
```

```
RBSFA`Private`ps[RBSFA`Private`t_, RBSFA`Private`tt_] :=
  RBSFA`Private`ps[RBSFA`Private`t, RBSFA`Private`tt] = - 1/(RBSFA`Private`t - RBSFA`Private`tt - i RBSFA`Private`ε)
      Inverse[IdentityMatrix[Length[RBSFA`Private`A[RBSFA`Private`tInit]]] -
          (2 RBSFA`Private`QuadMatrix[RBSFA`Private`t, RBSFA`Private`tt])/(RBSFA`Private`t - RBSFA`Private`tt - i RBSFA`Private`ε)].
        (RBSFA`Private`AInt[RBSFA`Private`t, RBSFA`Private`tt] -
          RBSFA`Private`PScorrectionInt[RBSFA`Private`t, RBSFA`Private`tt])
```

```
RBSFA`Private`pi
```

```
RBSFA`Private`pi[RBSFA`Private`p_, RBSFA`Private`t_, RBSFA`Private`tt_] :=
  RBSFA`Private`p + RBSFA`Private`A[RBSFA`Private`t] -
    RBSFA`Private`GAInt[RBSFA`Private`t, RBSFA`Private`tt].RBSFA`Private`p -
    RBSFA`Private`GAdotAInt[RBSFA`Private`t, RBSFA`Private`tt]
```

```
RBSFA`Private`S
```

```
RBSFA`Private`S[RBSFA`Private`t_, RBSFA`Private`tt_] := RBSFA`Private`simplifier[
    1/2 (Total[RBSFA`Private`ps[RBSFA`Private`t, RBSFA`Private`tt]^2] + RBSFA`Private`κ^2)
        (RBSFA`Private`t - RBSFA`Private`tt) + RBSFA`Private`ps[RBSFA`Private`t, RBSFA`Private`tt].
        RBSFA`Private`AInt[RBSFA`Private`t, RBSFA`Private`tt] +
      1/2 RBSFA`Private`A2Int[RBSFA`Private`t, RBSFA`Private`tt] -
      (RBSFA`Private`ps[RBSFA`Private`t, RBSFA`Private`tt].RBSFA`Private`QuadMatrix[RBSFA`Private`t,
            RBSFA`Private`tt].RBSFA`Private`ps[RBSFA`Private`t, RBSFA`Private`tt] +
        RBSFA`Private`ps[RBSFA`Private`t, RBSFA`Private`tt].
          RBSFA`Private`PScorrectionInt[RBSFA`Private`t, RBSFA`Private`tt] +
        RBSFA`Private`constCorrectionInt[RBSFA`Private`t, RBSFA`Private`tt])]
```

(abridged.)

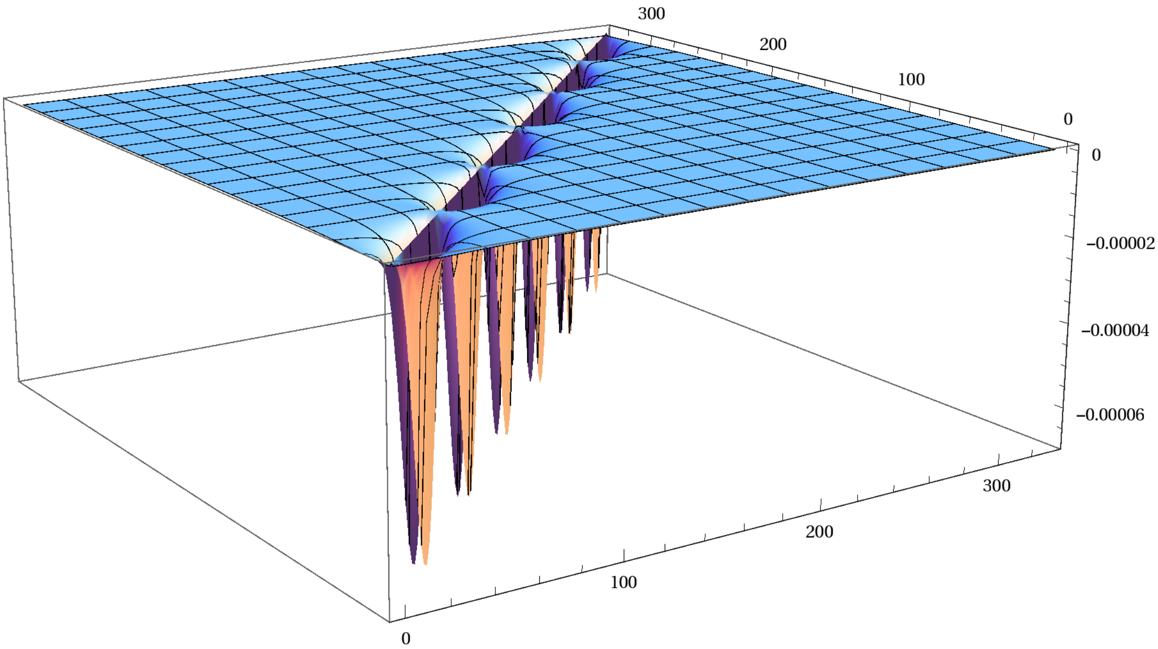
```
{{-0.0181258 - 0.0000997423 i, 0. + 0. i, 0. + 0. i}, {-0.018315 - 1.44441 i, 0. + 0. i, 0. + 0. i},
  {-0.0180896 - 5.34162 i, 0. + 0. i, 0. + 0. i}, {-0.0171312 - 10.575 i, 0. + 0. i, 0. + 0. i},
  {-0.0153823 - 15.8078 i, 0. + 0. i, 0. + 0. i}, {-0.013213 - 19.9497 i, 0. + 0. i, 0. + 0. i},
  {-0.0111113 - 22.4178 i, 0. + 0. i, 0. + 0. i}, {-0.0091716 - 23.1444 i, 0. + 0. i, 0. + 0. i},
  {-0.00705106 - 22.4188 i, 0. + 0. i, 0. + 0. i}, {-0.00454356 - 20.6872 i, 0. + 0. i, 0. + 0. i}}
```

- Setting Verbose→2 makes makeDipoleList output its key internal functions and shut down before the integration takes place. Its results can be caught as follows:

```
{A[t_], GA[t_], ps[t_, tt_], pi[p_, t_, tt_], S[t_, tt_], AInt[t_, tt_],
    A2Int[t_, tt_], GAInt[t_, tt_], GAdotAInt[t_, tt_], AdotGAInt[t_, tt_],
    GAIntInt[t_, tt_], PScorrectionInt[t_, tt_], constCorrectionInt[t_, tt_],
    GAIntdotGAIntInt[t_, tt_], QuadMatrix[t_, tt_], integrand[t_, τ_]} =
  makeDipoleList[VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}],
    FieldParameters → {F → 0.05, ω → 0.057}, Verbose → 2];
```

This then enables examination of e.g. the action:

```
Block[{ω = 0.057},
  Plot3D[
    Im [S[t, tt]]
    , {t, 0, 3 2π/ω}, {tt, 0, 3 2π/ω}
    , PlotRange → Full, ImageSize → 600, PlotTheme → "Classic", PlotPoints → 100
  ]
]
```



See the implementation notes in the code for makeDipoleList for further definitions of what each term entails.

- Setting Verbose→3 makes makeDipoleList return as pure functions the prefactor and action of the SFA integrand, as functions of the recombination time *t* and the ionization time *t*' = tt, which can then be used for e.g. quantum orbit calculations.

  Note that the precise name of this setting is subject to change in future versions.

```
Block[{prefactor, S},
  {prefactor, S} = makeDipoleList[VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}],
     FieldParameters → {F → 0.05, ω → 0.057}, Verbose → 3];
  Simplify[prefactor[t, tt] Exp[-i S[t, tt]]]
]
```

$$\left\{-\left(\left((0.+10.2129\,i)\, e^{(0.-0.192367\,i)\,t+(0.+0.192367\,i)\,tt-\frac{1}{2}i\,(t-tt)\left(1.+\frac{(15.3894\,\mathrm{Cos}[0.057\,t]-15.3894\,\mathrm{Cos}[0.057\,tt])^2}{((0.-0.1\,i)+t-1.\,tt)^2}\right)+\frac{i\,(15.3894\,\mathrm{Cos}[0.057\,t]-15.3894\,\mathrm{Cos}[0.057\,tt])}{(0.-0.1\,i)+t-1.\,tt}} \left(\frac{1}{0.1+i\,(t-tt)}\right)^{3/2} \mathrm{Cos}[0.057\,tt] \left(\frac{15.3894\,\mathrm{Cos}[0.057\,t]-15.3894\,\mathrm{Cos}[0.057\,tt]}{(0.-0.1\,i)+t-1.\,tt}+0.877193\,\mathrm{Sin}[0.057\,t]\right) \left(\frac{15.3894\,\mathrm{Cos}[0.057\,t]-15.3894\,\mathrm{Cos}[0.057\,tt]}{(0.-0.1\,i)+t-1.\,tt}+0.877193\,\mathrm{Sin}[0.057\,tt]\right)\right)\Big/ \left(\left(1.+\left((15.3894\,\mathrm{Cos}[0.057\,t]-15.3894\,\mathrm{Cos}[0.057\,tt])/((0.-0.1\,i)+t-1.\,tt)+0.877193\,\mathrm{Sin}[0.057\,t]\right)^2\right)^3 \left(1.+\left((15.3894\,\mathrm{Cos}[0.057\,t]-15.3894\,\mathrm{Cos}[0.057\,tt])/((0.-0.1\,i)+t-1.\,tt)+0.877193\,\mathrm{Sin}[0.057\,tt]\right)^2\right)^3\right)\right),\ 0,\ 0\right\}$$

If the action is required in symbolic form with respect to some parameter, the option CheckNumericFields can be used to turn off the usual check for numeric-valued field functions.

```
?CheckNumericFields
```

CheckNumericFields is an option for makeDipoleList which specifies whether to check for numeric values of A[t] and GA[t] for numeric t.

```
Block[{prefactor, S},
  {prefactor, S} = makeDipoleList[
     VectorPotential → Function[t, {F/ω Sin[ω t], 0, 0}], Verbose → 3, CheckNumericFields → False,
     CarrierFrequency → ω, IonizationPotential → Ip, ϵCorrection → ϵ];
  prefactor[t, tt] Exp[-ⅈ S[t, tt]]
]
```

$$\left\{-\left(\left(2048\, i\, e^{-i\left(-\frac{\left(-\frac{F\,\mathrm{Cos}[t\omega]}{\omega^2}+\frac{F\,\mathrm{Cos}[tt\omega]}{\omega^2}\right)^2}{t-tt-i\epsilon}+\frac{1}{2}(t-tt)\left(2\,Ip+\frac{\left(-\frac{F\,\mathrm{Cos}[t\omega]}{\omega^2}+\frac{F\,\mathrm{Cos}[tt\omega]}{\omega^2}\right)^2}{(t-tt-i\epsilon)^2}\right)+\frac{1}{2}\left(\frac{F^2\left(\frac{t}{2}-\frac{\mathrm{Sin}[2t\omega]}{4\omega}\right)}{\omega^2}-\frac{F^2\left(\frac{tt}{2}-\frac{\mathrm{Sin}[2tt\omega]}{4\omega}\right)}{\omega^2}\right)\right)}\, F\sqrt{Ip^{5/2}}\left(\frac{1}{i\,(t-tt)+\epsilon}\right)^{3/2}\mathrm{Conjugate}\left[\sqrt{Ip^{5/2}}\right]\mathrm{Cos}[tt\omega]\left(-\frac{-\frac{F\,\mathrm{Cos}[t\omega]}{\omega^2}+\frac{F\,\mathrm{Cos}[tt\omega]}{\omega^2}}{t-tt-i\epsilon}+\frac{F\,\mathrm{Sin}[t\omega]}{\omega}\right)\left(-\frac{-\frac{F\,\mathrm{Cos}[t\omega]}{\omega^2}+\frac{F\,\mathrm{Cos}[tt\omega]}{\omega^2}}{t-tt-i\epsilon}+\frac{F\,\mathrm{Sin}[tt\omega]}{\omega}\right)\right)\Big/\left(\sqrt{\pi}\left(2\,Ip+\left(-\frac{-\frac{F\,\mathrm{Cos}[t\omega]}{\omega^2}+\frac{F\,\mathrm{Cos}[tt\omega]}{\omega^2}}{t-tt-i\epsilon}+\frac{F\,\mathrm{Sin}[t\omega]}{\omega}\right)^2\right)^3\left(2\,Ip+\left(-\frac{-\frac{F\,\mathrm{Cos}[t\omega]}{\omega^2}+\frac{F\,\mathrm{Cos}[tt\omega]}{\omega^2}}{t-tt-i\epsilon}+\frac{F\,\mathrm{Sin}[tt\omega]}{\omega}\right)^2\right)^3\right)\right),\ 0,\ 0\right\}$$

It is also important to note, particularly if these functions are used for quantum orbits calculations, that the action does *not* include the Fourier-transform factor of $e^{i\Omega t}$, which affects both the phase and the amplitude of the harmonic dipole when evaluated at a complex saddle-point time $t = t_s$.

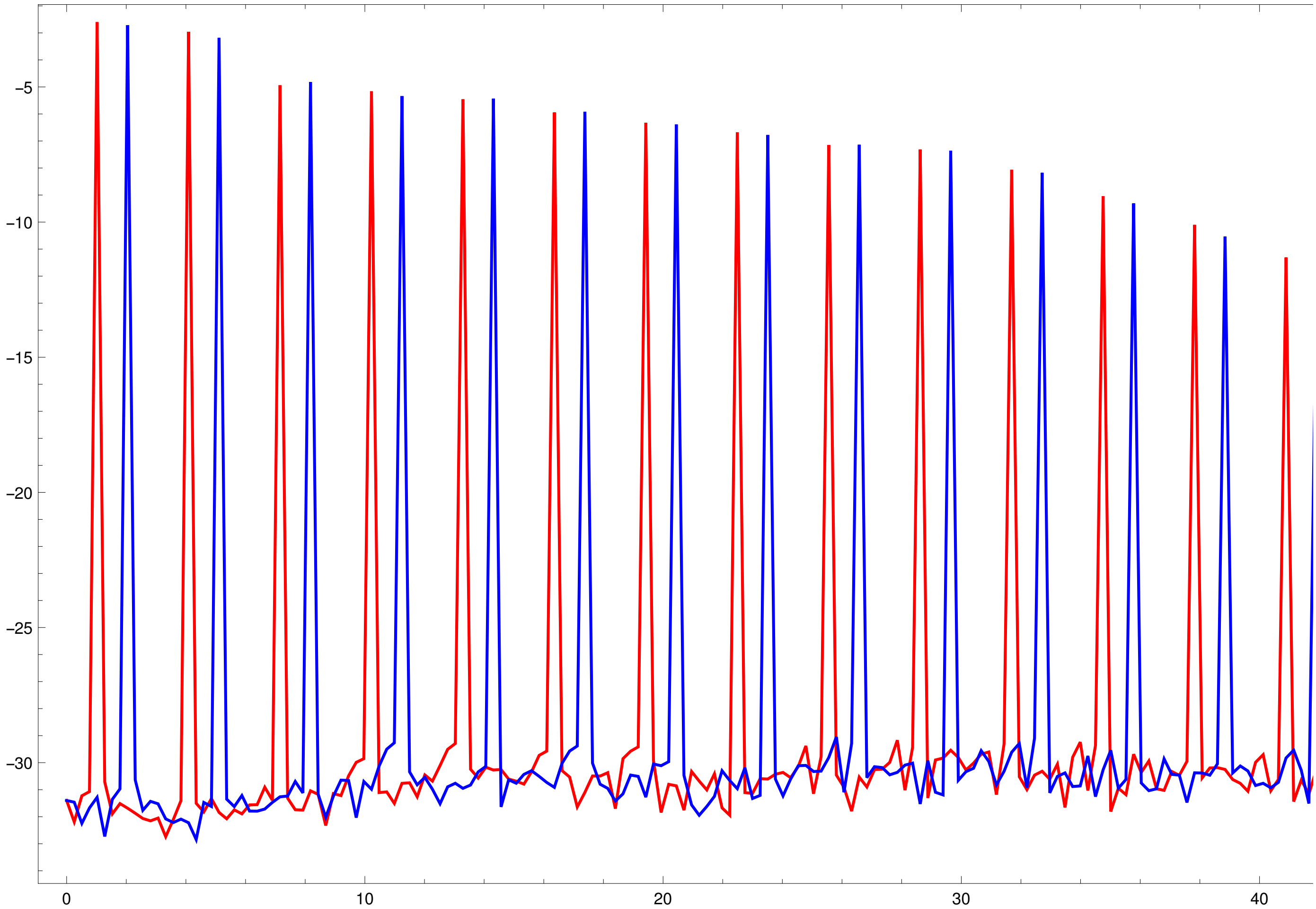
Bicircular fields

As a slightly less trivial example, consider a bicircular field: two counter-rotating, circularly polarized fields of different frequencies. The ‘standard’ case - as first demonstrated experimentally - has one field as the second harmonic of the fundamental, with both at equal intensities. The resultant harmonics appear at all integer orders except those divisible by three, with the $3n+1$ harmonics polarized as the fundamental, and the $3n-1$ harmonics polarized as the second-harmonic driver.

```
Quit

bicircularA[t_] := (F1/ω1 {Cos[t ω1] Sin[α], -Cos[α] Sin[t ω1]} + F2/ω2 {Cos[β] Cos[ω2 t], Sin[β] Sin[ω2 t]})
bicircularParameters = {F1 → 0.075, F2 → 0.075, α → 45 °, β → 45 °, ω1 → 45.6 / 800, ω2 → 45.6 / 400};
AbsoluteTiming [bicircularTest = makeDipoleList [VectorPotential → bicircularA,
        FieldParameters → bicircularParameters , TotalCycles → 4];]

{10.7277, Null}
```

The function biColorSpectrum takes the spectrum and plots it, separating the two circular polarizations into different colours.

```
biColorSpectrum [Most[bicircularTest]]
```



Bicircular fields with a sine-squared envelope

To benchmark the original calculations, we compared them with the output of full MCTDH calculations. Here we used a $\sin^2$ envelope as the TDSE numerics require a finite pulse; the calculations take correspondingly longer but they are still very manageable (two/three minutes per calculation for a fifteen-cycle pulse, resolving up to ~70 harmonics). One distinctive feature is that the harmonics near the cutoff are broader, because less cycles contribute to those energies.

```
bicircularEnvelopeA[t_] := cosPowerFlatTop[ω1, TotalCycles, 2][t]
    (F1/ω1 {Cos[t ω1] Sin[α], -Cos[α] Sin[t ω1]} + F2/ω2 {Cos[β] Cos[ω2 t], Sin[β] Sin[ω2 t]});
bicircularParameters = {F1 → 0.075, F2 → 0.075, α → 45 °, β → 45 °, ω1 → 45.6 / 800, ω2 → 45.6 / 400};
```

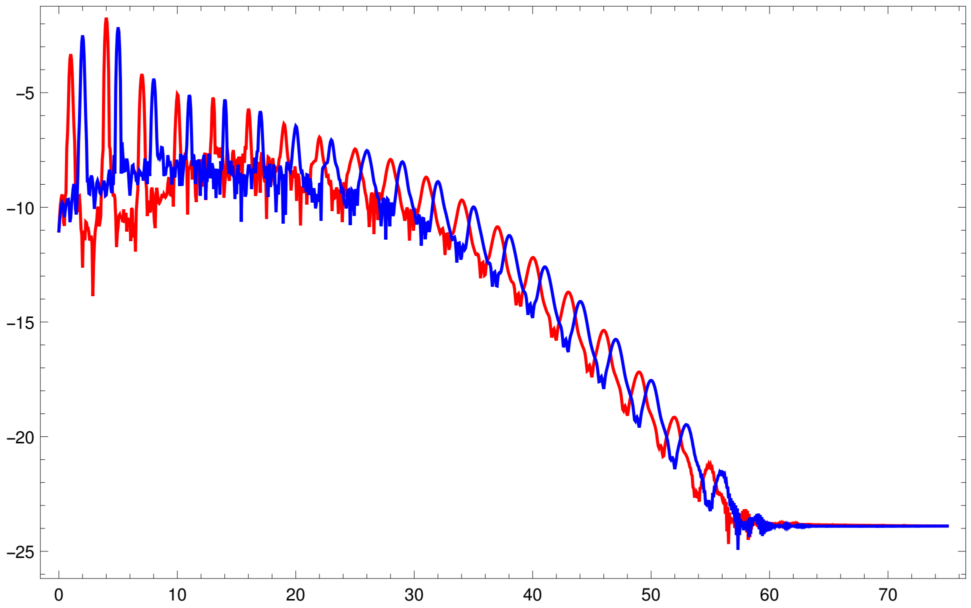
If (as in this case) the field depends on a number-of-cycles parameter, care must be taken that it matches the num option of the main call.

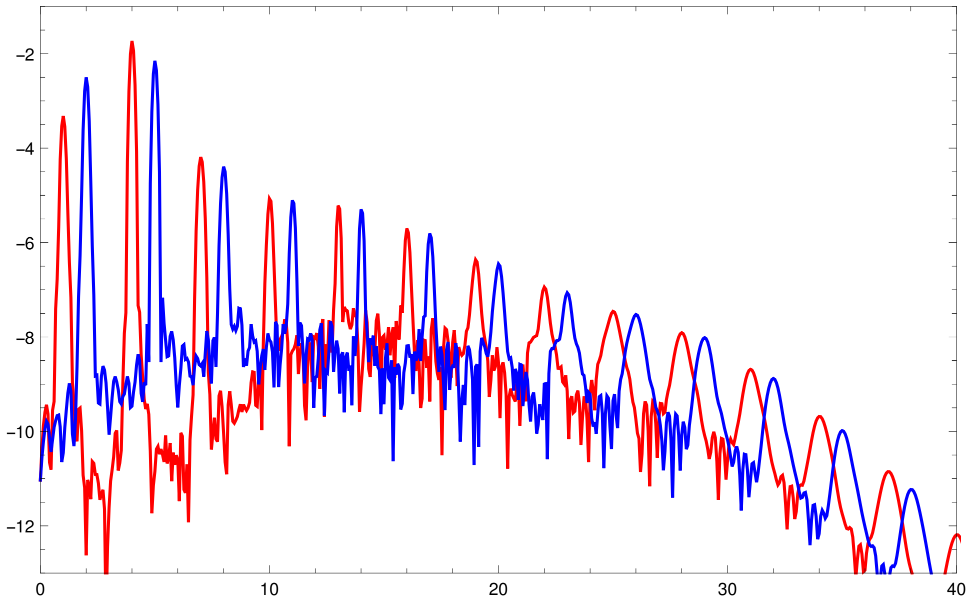
```
AbsoluteTiming[
  bicircularEnvelopeDipole = makeDipoleList[VectorPotential → bicircularEnvelopeA,
        FieldParameters → Join[bicircularParameters, {TotalCycles → 15}],
        PointsPerCycle → 150, TotalCycles → 15];
]
{173.565, Null}
```

```
{141.302, Null}
```

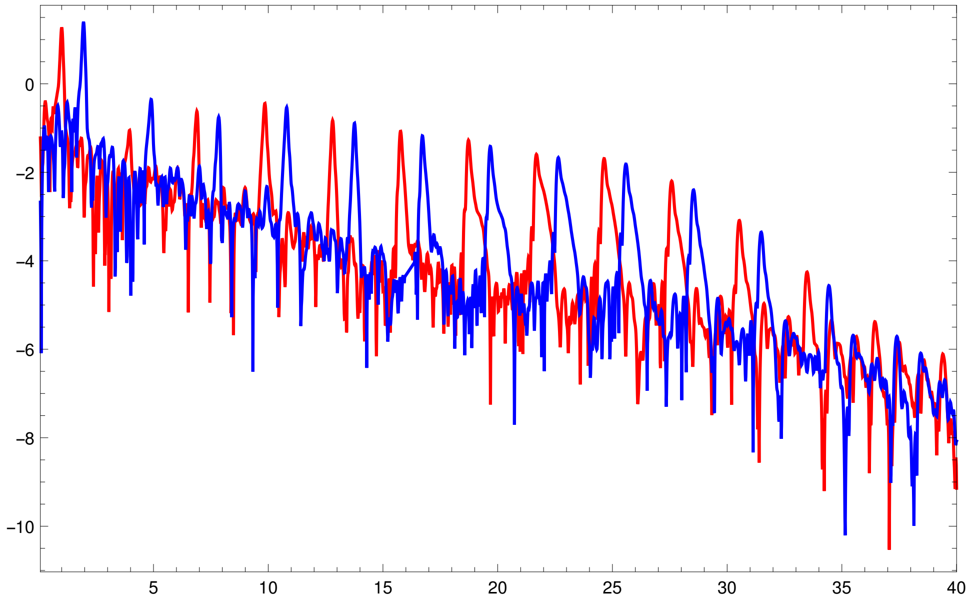
Plotting the spectrum, and a zoom at the plateau:

```
biColorSpectrum [bicircularEnvelopeDipole,
  PointsPerCycle→150, TotalCycles→15, ImageSize →500]
biColorSpectrum [bicircularEnvelopeDipole, PointsPerCycle→150,
  TotalCycles→15, ImageSize →500, PlotRange→{{0, 40}, {-13, -1}}]
```





The comparable MCTDH spectrum, for identical conditions, looks like this:



Original RB-SFA: ‘rotating’ bicircular fields

Calculation

Here the fundamental laser driver has been set at an elliptical polarization (as in the original experiment, A. Fleischer et al., *Nature Photon.* **8**, 543 (2014)), which helps investigate the spin-angular-momentum conservation properties of HHG. In the model proposed in the original paper (*Phys. Rev. A* **90**, 043829 (2014)), the photon model is validated by splitting the elliptical field itself into two circular components, which can then be tuned independently:

```
rotatingBicircularA[t_] := envelope[t] (F2/ω2 {Cos[β] Cos[ω2 t - ϕ1], Sin[β] Sin[ω2 t - ϕ1]} +
    F1/√2 (1/ω1 Cos[α - π/4] {Cos[ω1 t + ϕ1], -Sin[ω1 t + ϕ1]} +
      1/((1 + δ) ω1) Sin[α - π/4] {Cos[(1 + δ) ω1 t - ϕ1 + ϕ2], +Sin[(1 + δ) ω1 t - ϕ1 + ϕ2]}));
DistributeDefinitions["RBSFA`"];
directory= FileNameJoin [{NotebookDirectory[], "Temp  Data"}];
filename [δ_] :=
    FileNameJoin [{directory, "data 25.09 detuning scan at δ=" <> ToString[δ] <> ".txt"}];
Length[δRange = Range[0, 0.25, 0.001]]
251
```

To test the validity of the photon model, we ran a scan over the detuning δ, using the calculation below.

```
DateString[]
Print["Total = ", Length[δRange], " points at ~230s/point will be done at approximately  ",
  DateString[AbsoluteTime [] + Length[δRange] * 230. / 7], "."]
ParallelTable[
    Print[AbsoluteTiming [
        makeDipoleList [
            VectorPotential → rotatingBicircularA,
            FieldParameters → {α → 35 °, β → 45 °, F1 → 0.075, F2 → 0.075, ω1 → 0.057,
                ω2 → 1.95 × 0.057, ϕ1 → 0, ϕ2 → 0, envelope → flatTopEnvelope[ω1, 26, 3]},
            CarrierFrequency → 0.057, TotalCycles → 26, PointsPerCycle → 115,
            nGate → 1.8, PointNumberCorrection → 1, Preintegrals → "Numeric ",
            ReportingFunction → Function[Write[filename [δ], #]]
          ];]];
    Print[DateString[]];
    , {δ, δRange}];
DateString[]
NotebookSave[]
```

Total time 2h 32min. (Desktop machine with 8-thread, 4-core Intel i7-3770 CPU at 3.40GHz, 16GB RAB, running 7 *Mathematica* kernels in parallel.)
Expand this cell to see the calculation log.

The results can be pulled in from the files using this:

```
Do[detunedDipole[δ] = ReadList[filename [δ]], {δ, δRange}]
```

Or saved into a single location using this:

```
Save[FileNameJoin [{NotebookDirectory[], "Detuning scan collected data.txt"}], detunedDipole]
```

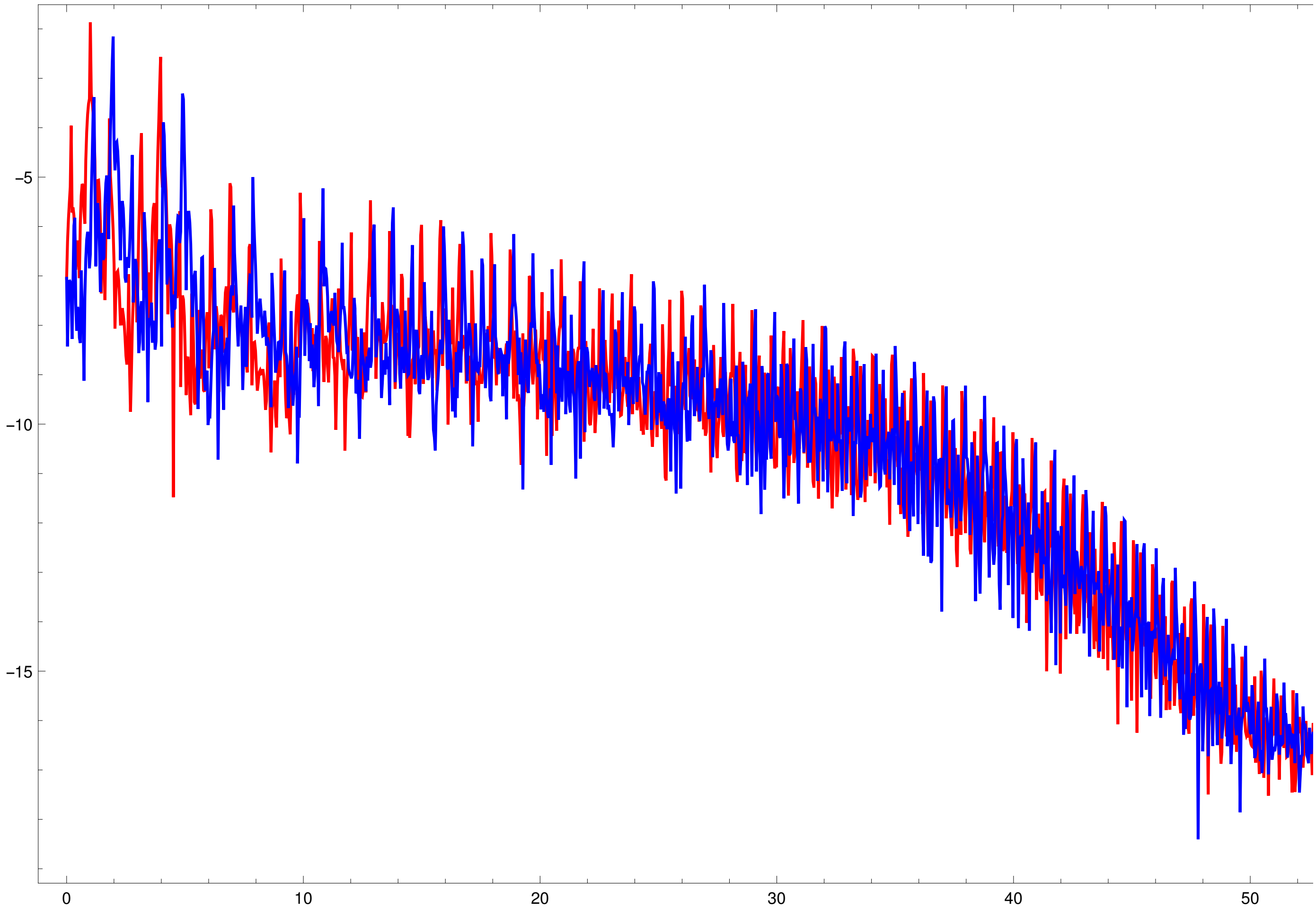
```
DumpSave [
    FileNameJoin [{NotebookDirectory[], "Detuning scan collected data.mx "}], detunedDipole];
```

and pulled in from the single location using this:

```
<< (FileNameJoin [{NotebookDirectory[], "Detuning scan collected data.txt"}]);
```

A sample spectrum looks like this:

```
biColorSpectrum [detunedDipole[0.001 RandomInteger [{0, 1000}]],
  CarrierFrequency→45.6 / 800, TotalCycles→3, PointsPerCycle→115]
```



Plots from the original paper

The plots from the original paper were produced using the code below. For simplicity we pre-define an interpolation function.

```
conditions: = Sequence[CarrierFrequency→45.6 / 800, TotalCycles→26, PointsPerCycle→115]
```

```
Remove [detuningInterpolation]
With[{length = Length[getSpectrum [detunedDipole[0.], Polarization→ {1, ⅈ}]]},
  AbsoluteTiming [
    Table[
        detuningInterpolation[ϵ] = Interpolation[
            Flatten[Table[
                {{
                        harmonicOrderAxis [TargetLength → length, conditions],
                        Table[δ, {length}]
                      }ᵀ,
                    Log[10, getSpectrum [detunedDipole[δ], Polarization→ {1, ϵ ⅈ}]]
                  }ᵀ
                , {δ, δRange}], 1]]
        , {ϵ, {1, -1}}];
  ]
]
{2.99829, Null}
```

Some plotting admin:

```
CMRmap  = Function[x, Blend[{■, ■, ■, ■, ■, ■, ■, ■, □}, x]];
CMRwithMin[minIn_ , minOut_: 1./9] :=
  Function[x, CMRmap [If[x < minIn , (minOut/minIn) x, minOut + (1 - minOut ) (x - minIn)/(1 - minIn)]]]
min  = 6.×10^-9;
max  = 5.×10^-7;
colorfunction= CMRwithMin[min /max ];
HOTicks[ϵ_] :=
  ({#, If [ϵ == 1, Style[#, Black], ""], {0.02, 0}, {Thickness[0.005], Gray}} & /@Range[12, 18, 1]) ~
    Join~ ({#, "", {0.01, 0}, {Thickness[0.004], Gray}} & /@Range[11 + 1/2, 18 + 1/4, 1/4])
downTicks= {{0, Style[0, Black], 0}, {0.25, Style[0.25, Black], 0}} ~Join~
      ({#, Style[#, Black], {0.015, 0}, {Thickness[0.005], Gray}} & /@Range[0.05, 0.20, 0.05]) ~
      Join~ ({#, "", {0.01, 0}, {Thickness[0.004], Gray}} & /@Range[0.01, 0.24, 0.01]);
upTicks= ({#, "", {0.015, 0}, {Thickness[0.005], Gray}} & /@Range[0.05, 0.20, 0.05]) ~
      Join~ ({#, "", {0.01, 0}, {Thickness[0.004], Gray}} & /@Range[0.01, 0.24, 0.01]);
```

The plot itself:

```
Row[Table[
    splittingsScan[ϵ] = RegionPlot[
        True
        , {δ, 0, 0.25}, {HO, 11.25, 18.5}
        , AspectRatio→1.2
        , PlotRangePadding→None
        , ImagePadding →1 {{35+15ϵ, 20}, {70, 6}}
        , ImageSize →{Automatic , 550}
        , PlotPoints→600
        , FrameStyle →Automatic
        , FrameLabel →
          {Style["ω'/ω-1", Black, 12], If[ϵ == 1, Style["Harmonic  Order", Black, 16], ""]}
        , ColorFunctionScaling→False
        , FrameTicks →{{HOTicks[1], HOTicks[-1]}, {downTicks, upTicks}}
        , ColorFunction→Function[{δ, HO}, colorfunction[10^detuningInterpolation[ϵ][HO, δ]/max]]
        , PlotLabel→
          Style[StringJoin[ϵ /. {1→"Right", -1→"Left"}, "-circular harmonics "], Black, 16]
      ]
    , {ϵ, {1, -1}}]
]
```

(Removed to keep file size low.)