Axion Search Simulation

Sam Hindley, University of Liverpool

Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

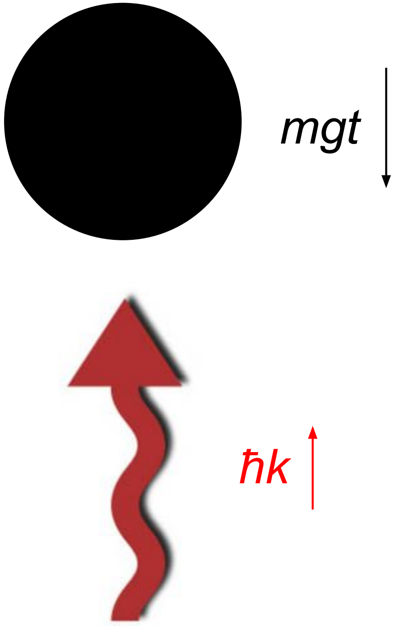
Overview

- Motivate axion search in MAGIS
- Schematic of software
- Discussion of software
- Results and Conclusions

Sam Hindley
sam.hindley@liverpool.ac.uk

Axion Search in MAGIS

- Axion signal thought to manifest itself as a “dark” magnetic field
- Couples to atom spins and produces anomalous magnetic coupling
- Unlike ultralight scalar DM search, freefall is not necessary



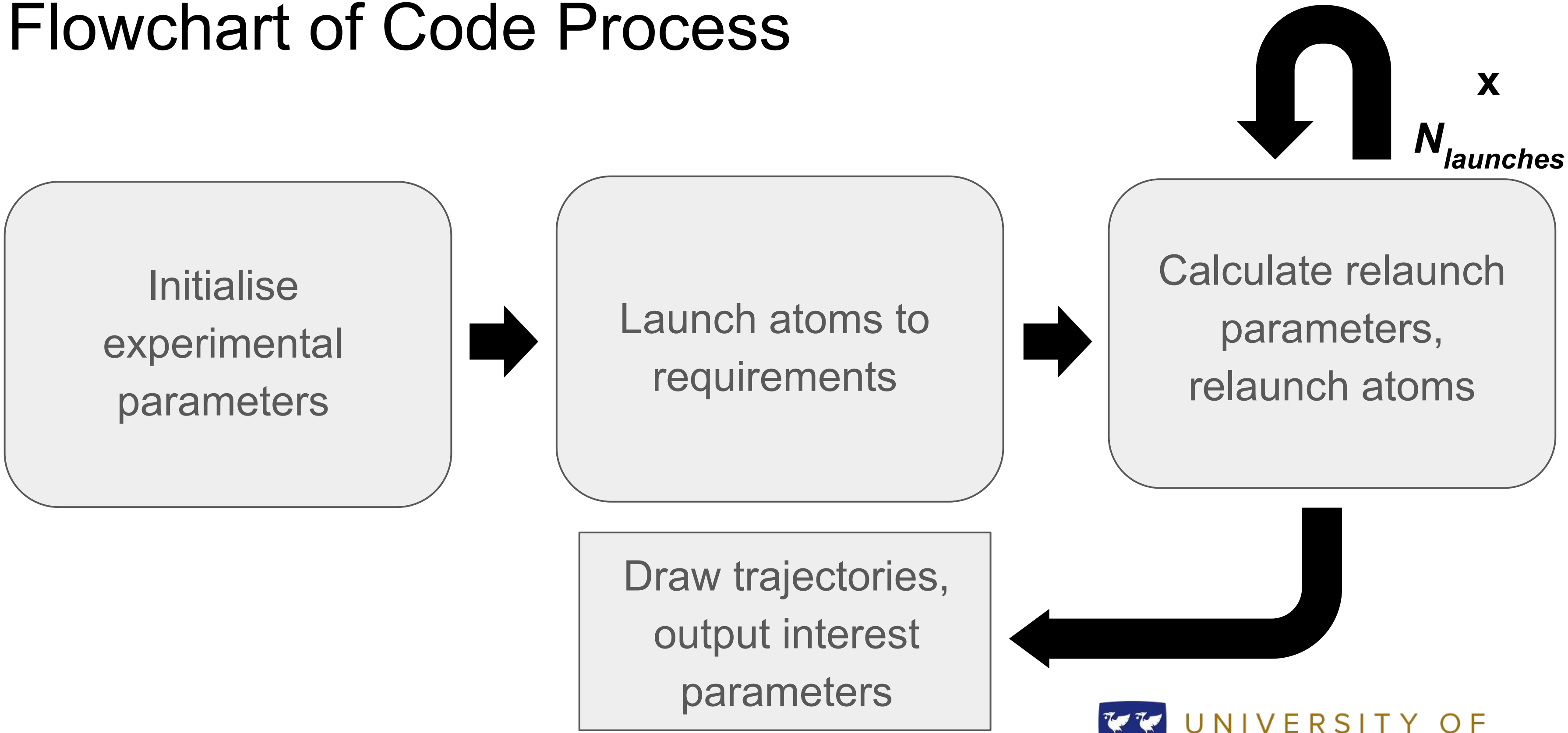
Sam Hindley
sam.hindley@liverpool.ac.uk
UNIVERSITY OF LIVERPOOL

Simulation Requirements

- Keep atoms within magnetically-shielded region
- Get axion sensitivity by maintaining opposed spins for greater than 1 s
  - Wrote a script to do this, uses standard Mathematica functions
- Store all relevant parameters at each timestep

Sam Hindley
sam.hindley@liverpool.ac.uk

Flowchart of Code Process



Sam Hindley
sam.hindley@liverpool.ac.uk
UNIVERSITY OF LIVERPOOL

Simulate Atom Cloud & Define Initial Parameters

Atom Ballistic Trajectory Simulation

Start with one atom at z = 0. It falls under gravity and is kicked back up by a photon that gives it $\hbar\kappa$ of momentum. Use LMT lasers as a model.

```
In[1]:= (* define basic physical parameters *)
z = 0; (* atom position, m *)
u = 0; (* initial velocity, m *)
ħ = 1.0546 * 10^-34 ; (* J sec *)
m = 1.44 * 10^-25 ; (* kg *)
k = 2 * π / (698 * 10^-9) ; (* wavevector, m^-1 *)
g = -9.81; (* ms^-2 *)
cloudradius = 0.0018; (* m, that's 1.8mm *)
```

```
In[8]:= tend = 4000 * 10^-3; (* total simulated time, s *)
tstep = 30 * 10^-6; (* time resolution, s *)
simlength = Floor[tend / tstep] + 1;
minpulsespeed = (161 + 275) * 10^-9;

kickspeed = 300 * 10^-6;
(* round numbered pulse spacing faster than the minimum
 pulse spacing that will increase arm momentum *)
shieldedge = 0.2; (* height of the shield, in m *)
shieldbottom = 0;
(* bottom of shield defaults to the origin,
could hypothetically be moved *)
nkicks = 330; (* number of fast kicks during separation
 sequence *)
```

Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

Create Timebase & Launch

Setup upper arm - this one will be kicked faster initially to get the 5ℏκ separation:

```
uptlist1 = Table[0, simlength];
(* define empty tables for each of the physical quantities
 involved *)
upslist1 = Table[0, simlength];
upvlist1 = Table[0, simlength];
upplist1 = Table[0, simlength];
uppulselist = Table[0, simlength];
uppulselist[[1]] = ℏ * k; (* start from p = ℏk *)
For[i1 = 1, i1 < nkicks + 1, i1++,
  uppulselist[[(i1 * kickspeed / (tstep))]] = ℏ * k;] ;
 (* create a list of momentum kicks for the launch
 sequence *)
```

First simulation, no relaunch:

```
(* Loop upper arm *)
For[i = 1, i < simlength, i++,
 uptlist1[[i + 1]] = uptlist1[[i]] + tstep;
 upplist1[[i + 1]] = m * upvlist1[[i]];
 upvlist1[[i + 1]] = upvlist1[[i]] + g * tstep +
   (uppulselist[[i]] / m);
 upslist1[[i + 1]] = upslist1[[i]] + upvlist1[[i]] * tstep ]

(* Loop lower arm *)
For[i = 1, i < simlength, i++,
 lotlist1[[i + 1]] = lotlist1[[i]] + tstep;
 loplist1[[i + 1]] = m * lovlist1[[i]];
 lovlist1[[i + 1]] = lovlist1[[i]] + g * tstep +
   (lopulselist[[i]] / m);
 loslist1[[i + 1]] = loslist1[[i]] + lovlist1[[i]] * tstep ]
```

Sam Hindley
sam.hindley@liverpool.ac.uk

Create Timebase & Launch

Setup upper arm - this one will be kicked faster initially to get the 5ℏκ separation:

```
uptlist1 = Table[0, simlength];
(* define empty tables for each of the physical quantities
 involved *)
upslist1 = Table[0, simlength];
upvlist1 = Table[0, simlength];
upplist1 = Table[0, simlength];
uppulselist = Table[0, simlength];
uppulselist[[1]] = ℏ * k; (* start from p = ℏk *)
For[i1 = 1, i1 < nkicks + 1, i1++,
  uppulselist[[(i1 * kickspeed / (tstep))]] = ℏ * k;] ;
 (* create a list of momentum kicks for the launch
 sequence *)
```

First simulation, no relaunch:

```
(* Loop upper arm *)
For[i = 1, i < simlength, i++,
 uptlist1[[i + 1]] = uptlist1[[i]] + tstep;
 upplist1[[i + 1]] = m * upvlist1[[i]];
 upvlist1[[i + 1]] = upvlist1[[i]] + g * tstep +
    (uppulselist[[i]] / m);
 upslist1[[i + 1]] = upslist1[[i]] + upvlist1[[i]] * tstep ]

(* Loop lower arm *)
For[i = 1, i < simlength, i++,
 lotlist1[[i + 1]] = lotlist1[[i]] + tstep;
 loplist1[[i + 1]] = m * lovlist1[[i]];
 lovlist1[[i + 1]] = lovlist1[[i]] + g * tstep +
    (lopulselist[[i]] / m);
 loslist1[[i + 1]] = loslist1[[i]] + lovlist1[[i]] * tstep ]
```

Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

Create Timebase & Launch

Setup upper arm - this one will be kicked faster initially to get the 5ħκ separation:

```
uptlist1 = Table[0, simlength];
(* define empty tables for each of the physical quantities
 involved *)
upslist1 = Table[0, simlength];
upvlist1 = Table[0, simlength];
upplist1 = Table[0, simlength];
uppulselist = Table[0, simlength];
uppulselist[[1]] = ħ * k; (* start from p = ħk *)
For[i1 = 1, i1 < nkicks + 1, i1++,
  uppulselist[[(i1 * kickspeed / (tstep))]] = ħ * k;] ;
 (* create a list of momentum kicks for the launch
 sequence *)
```

First simulation, no relaunch:

```
(* Loop upper arm *)
For[i = 1, i < simlength, i++,
 uptlist1[[i + 1]] = uptlist1[[i]] + tstep;
 upplist1[[i + 1]] = m * upvlist1[[i]];
 upvlist1[[i + 1]] = upvlist1[[i]] + g * tstep +
   (uppulselist[[i]] / m);
 upslist1[[i + 1]] = upslist1[[i]] + upvlist1[[i]] * tstep ]

(* Loop lower arm *)
For[i = 1, i < simlength, i++,
 lotlist1[[i + 1]] = lotlist1[[i]] + tstep;
 loplist1[[i + 1]] = m * lovlist1[[i]];
 lovlist1[[i + 1]] = lovlist1[[i]] + g * tstep +
   (lopulselist[[i]] / m);
 loslist1[[i + 1]] = loslist1[[i]] + lovlist1[[i]] * tstep ]
```

Sam Hindley
sam.hindley@liverpool.ac.uk

Determine Relaunch Parameters & Iterate

```
(* make a fit line for the falling region *)

upplistpeak = Max[upplist1];
upplistpeaksite =
  Flatten[Position[upplist1, upplistpeak]][[1]];
upplistpeaktime = upplistpeaksite * tstep;

upfallregionstart = upplistpeaktime;
upfallregionend = upplistpeaktime + (kickspeed * 600);
upfallregstartstep = Floor[upfallregionstart / tstep];
upfallregendstep = Floor[upfallregionend / tstep];

upfallregsites =
  Table[i, {i, upfallregstartstep, upfallregendstep}];
upfallregpvalues = upplist1[[upfallregsites]] / (ħ * k);
upfallregtvalues = uptlist1[[upfallregsites]];
upfallregcoords =
  Transpose[{upfallregpvalues, upfallregtvalues}];
upfallregfit = LinearModelFit[upfallregcoords, x, x];
```

```
(* Find the point where the momentum minimum is equal
 to the peak momentum,
and calculate how many pulses would be needed to cancel
 it*)

uppeakmomentum = Max[upplist1];
uppeakmomentumkicks = Floor[uppeakmomentum / (ħ * k)];
upcancelkicks = Abs[Ceiling[uppeakmomentumkicks / pmodfactor]];
uprelaunchtime = upfallregfit[(-1 * uppeakmomentumkicks)];
uprelaunchtimestep = Floor[uprelaunchtime / tstep];
upnrelaunch1 = upcancelkicks + nkicks;

For[i1 = 1, i1 < nkicks + 1, i1++,
  uppulselist[[(i1 * kickspeed / (tstep))]] = ħ * k;] ;
 (* create a list of momentum kicks for the launch sequence *)
For[i1 = 1, i1 < upnrelaunch1 + 1, i1++,
  uppulselist[[
     (uprelaunchtimestep + (i1 * kickspeed / (tstep)))]] =
    ħ * k;] ;(* create a list of momentum kicks for the
 relaunch sequence *)
```

Sam Hindley
sam.hindley@liverpool.ac.uk

Determine Relaunch Parameters & Iterate

```
(* make a fit line for the falling region *)

upplistpeak = Max[upplist1];
upplistpeaksite =
  Flatten[Position[upplist1, upplistpeak]][[1]];
upplistpeaktime = upplistpeaksite * tstep;

upfallregionstart = upplistpeaktime;
upfallregionend = upplistpeaktime + (kickspeed * 600);
upfallregstartstep = Floor[upfallregionstart / tstep];
upfallregendstep = Floor[upfallregionend / tstep];

upfallregsites =
  Table[i, {i, upfallregstartstep, upfallregendstep}];
upfallregpvalues = upplist1[[upfallregsites]] / (ħ * k);
upfallregtvalues = uptlist1[[upfallregsites]];
upfallregcoords =
  Transpose[{upfallregpvalues, upfallregtvalues}];
upfallregfit = LinearModelFit[upfallregcoords, x, x];
```

```
(* Find the point where the momentum minimum is equal
 to the peak momentum,
and calculate how many pulses would be needed to cancel
 it*)

uppeakmomentum = Max[upplist1];
uppeakmomentumkicks = Floor[uppeakmomentum / (ħ * k)];
upcancelkicks = Abs[Ceiling[uppeakmomentumkicks / pmodfactor]];
uprelaunchtime = upfallregfit[(-1 * uppeakmomentumkicks)];
uprelaunchtimestep = Floor[uprelaunchtime / tstep];
upnrelaunch1 = upcancelkicks + nkicks;

For[i1 = 1, i1 < nkicks + 1, i1++,
  uppulselist[[(i1 * kickspeed / (tstep))]] = ħ * k;] ;
  (* create a list of momentum kicks for the launch sequence *)
For[i1 = 1, i1 < upnrelaunch1 + 1, i1++,
  uppulselist[[
     (uprelaunchtimestep + (i1 * kickspeed / (tstep)))]] =
    ħ * k;] ;(* create a list of momentum kicks for the
 relaunch sequence *)
```

Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

Determine Relaunch Parameters & Iterate

```
(* make a fit line for the falling region *)

upplistpeak = Max[upplist1];
upplistpeaksite =
  Flatten[Position[upplist1, upplistpeak]][[1]];
upplistpeaktime = upplistpeaksite * tstep;

upfallregionstart = upplistpeaktime;
upfallregionend = upplistpeaktime + (kickspeed * 600);
upfallregstartstep = Floor[upfallregionstart / tstep];
upfallregendstep = Floor[upfallregionend / tstep];

upfallregsites =
  Table[i, {i, upfallregstartstep, upfallregendstep}];
upfallregpvalues = upplist1[[upfallregsites]] / (ħ * k);
upfallregtvalues = uptlist1[[upfallregsites]];
upfallregcoords =
  Transpose[{upfallregpvalues, upfallregtvalues}];
upfallregfit = LinearModelFit[upfallregcoords, x, x];
```

```
(* Find the point where the momentum minimum is equal
 to the peak momentum,
and calculate how many pulses would be needed to cancel
 it*)

uppeakmomentum = Max[upplist1];
uppeakmomentumkicks = Floor[uppeakmomentum / (ħ * k)];
upcancelkicks = Abs[Ceiling[uppeakmomentumkicks / pmodfactor]];
uprelaunchtime = upfallregfit[(-1 * uppeakmomentumkicks)];
uprelaunchtimestep = Floor[uprelaunchtime / tstep];
upnrelaunch1 = upcancelkicks + nkicks;
```

```
For[i1 = 1, i1 < nkicks + 1, i1++,
  uppulselist[[(i1 * kickspeed / (tstep))]] = ħ * k;] ;
 (* create a list of momentum kicks for the launch sequence *)
For[i1 = 1, i1 < upnrelaunch1 + 1, i1++,
  uppulselist[[
     (uprelaunchtimestep + (i1 * kickspeed / (tstep)))]] =
    ħ * k;] ;(* create a list of momentum kicks for the
 relaunch sequence *)
```

Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

Determine Relaunch Parameters & Iterate

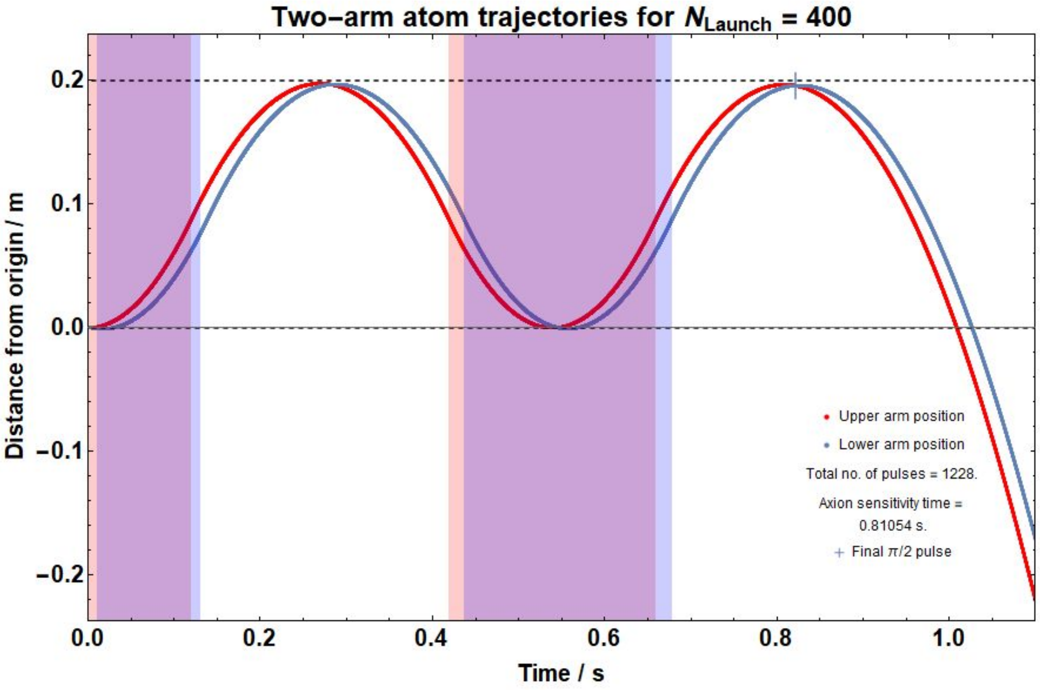
```
(* make a fit line for the falling region *)

upplistpeak = Max[upplist1];
upplistpeaksite =
  Flatten[Position[upplist1, upplistpeak]][[1]];
upplistpeaktime = upplistpeaksite * tstep;

upfallregionstart = upplistpeaktime;
upfallregionend = upplistpeaktime + (kickspeed * 600);
upfallregstartstep = Floor[upfallregionstart / tstep];
upfallregendstep = Floor[upfallregionend / tstep];

upfallregsites =
  Table[i, {i, upfallregstartstep, upfallregendstep}];
upfallregpvalues = upplist1[[upfallregsites]] / (ħ * k);
upfallregtvalues = uptlist1[[upfallregsites]];
upfallregcoords =
  Transpose[{upfallregpvalues, upfallregtvalues}];
upfallregfit = LinearModelFit[upfallregcoords, x, x];
```

```
(* Find the point where the momentum minimum is equal
 to the peak momentum,
and calculate how many pulses would be needed to cancel
 it*)

uppeakmomentum = Max[upplist1];
uppeakmomentumkicks = Floor[uppeakmomentum / (ħ * k)];
upcancelkicks = Abs[Ceiling[uppeakmomentumkicks / pmodfactor]];
uprelaunchtime = upfallregfit[(-1 * uppeakmomentumkicks)];
uprelaunchtimestep = Floor[uprelaunchtime / tstep];
upnrelaunch1 = upcancelkicks + nkicks;

For[i1 = 1, i1 < nkicks + 1, i1++,
  uppulselist[[(i1 * kickspeed / (tstep))]] = ħ * k;] ;
 (* create a list of momentum kicks for the launch sequence *)
For[i1 = 1, i1 < upnrelaunch1 + 1, i1++,
  uppulselist[[
     (uprelaunchtimestep + (i1 * kickspeed / (tstep)))]] =
    ħ * k;] ;(* create a list of momentum kicks for the
 relaunch sequence *)
```

Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

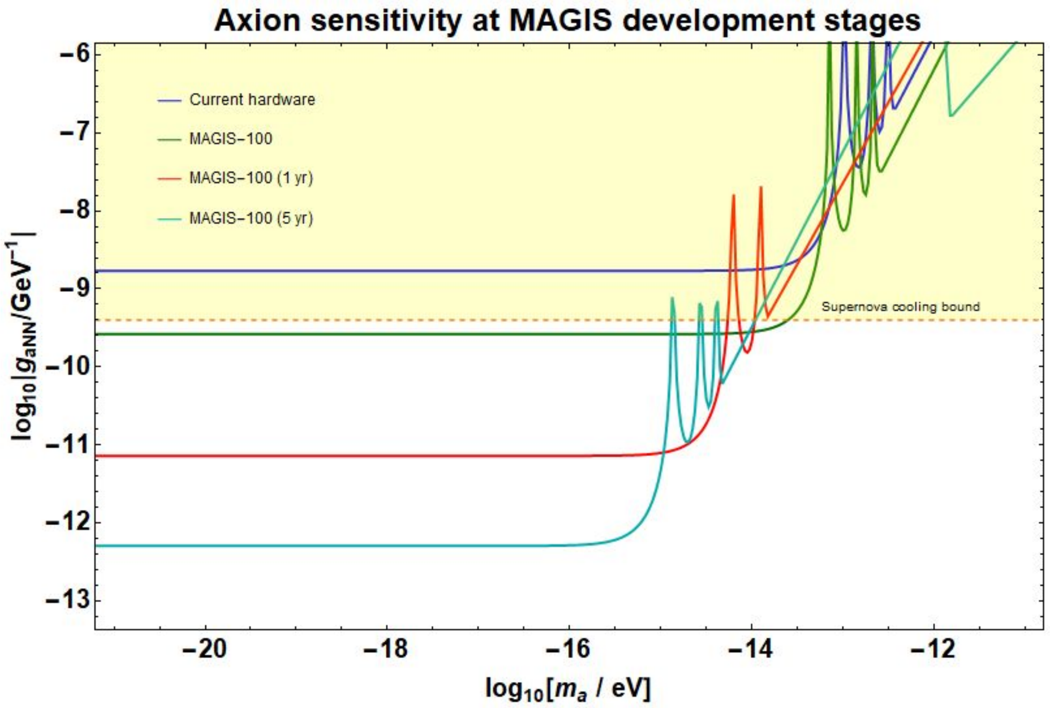
Print Trajectories & Details



Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

Sensitivity Plots



Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

Summary

- Software simulates two arms of an atom interferometer suspended against gravity inside a shield of variable size
- All physical quantities are stored at each timestep, trajectories can be plotted
- Interrogation time and final state atom count give sensitivities

	Shield Size	Max. Pulses	Smallest Sensitive g_{aNN}
Existing Parameters	4×10^{-3} m*	72	1.706×10^{-9}
MAGIS Parameters	8×10^{-3} m*	100	2.631×10^{-10}
Target Parameters	2×10^{-1} m	1000	7.215×10^{-12}
Long-term Parameters	1 m	5000	5.051×10^{-13}

Sam Hindley
sam.hindley@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL