7-19-2018

Detecting Rydberg Interactions With Controlled Ionization

Lauren Yoast
Ursinus College, layoast@ursinus.edu

Detecting Rydberg Interactions with Controlled Ionization

by
Lauren Yoast

Department of Physics and Astronomy
Ursinus College

July 20, 2018

Abstract

Rydberg atoms, which have a highly excited outer electron, are easily manipulated by electric fields. Using a magneto-optical trap, we cool Rubidium atoms to a few hundred millionths of a Kelvin above absolute zero and then excite to Rydberg states. Our first project looks at the dipole-dipole interactions of two atoms starting in the 33p state and ending in the 34s and 33s states. The standard technique is to apply an increasing electric field that ionizes the Rydberg electron and sends it to a detector, but unfortunately the signals overlap. A genetic algorithm is used to separate the signals by controlling the ionization pathway.

In order to understand how a Rydberg atom ionizes we need to build a model. The current model, which uses a semi-empirical formula, neglects important quantum phase information. In our second project we are building a computational model that includes continuum states along with discrete states. We present progress on a new model which takes into account the differences normalization for continuum and discrete states.

Mathematica proved to be an extremely helpful resource in both of these projects. Scans from a genetic algorithm were imported into a mathematica notebook so that graphs before and after could be seen. Voltage scan data was also imported into mathematica where it could be calibrated and graphed. Mathematica was also used heavily in trying to build a new model of ionization. We graphed radial wave functions and calculated matrix elements in order to try and understand the current model. From there, we started trying to find a way to include continuum states in our calculations.

Table of Contents

Chapter 1: Getting to Know Mathematica

Time Evolution of a Simple Pendulum

Computing the time evolution of a simple pendulum using the Euler method.

Setting Initial Values

In[∘]:=
```
theta = 30
theta = theta * π / 180; (*converting angle to radians*)
omega = 0; (*initial velocity*)
tau = 0.01; (*time step*)
gOverL = 1.0;
timeOld = -1.0;
irev = 0; (*used to count number of reversals*)
nstep = 3000; (*number of time steps*)
time = 0;
```

Out[∘]= 30

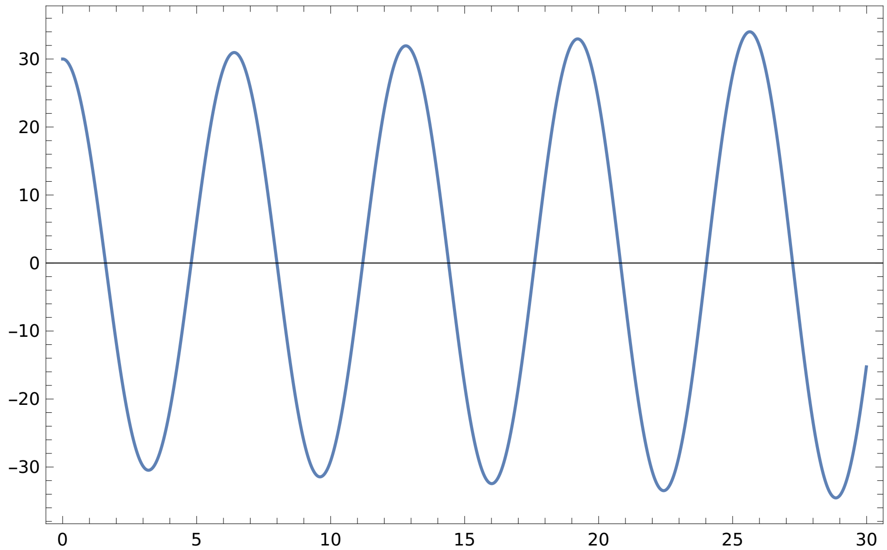
Motion of the Pendulum

In[∘]:=
```
data = {};
For[i = 1, i ≤ nstep, i++,
  AppendTo[data, {time, theta * 180 / π}];
  (*recording time and angle as points in data*)
  accel = -gOverL * Sin[theta]; (*gravitational acceleration*)
  thetaOld = theta;
  theta = theta + tau * omega; (*euler method*)
  omega = omega + tau * accel; (*euler method*)
  time = time + tau;
 ];
```

Plot of Pendulum Motion

In[•]:=
```
ListLinePlot[data, Joined → True, Frame → True]
```

Out[•]=



Random Atom Positions Inside of a Circle

Randomly plotting a given number of atoms evenly inside a circle.

Reading in List from Text Editor

```
(*partition in desired number of coordniates*)
data = Partition[BinaryReadList[
    "/home/layoast/multivac/source/cppExercise/test.dat", "Real64"], 2];
```

Changing Coordinates to Graph Correctly

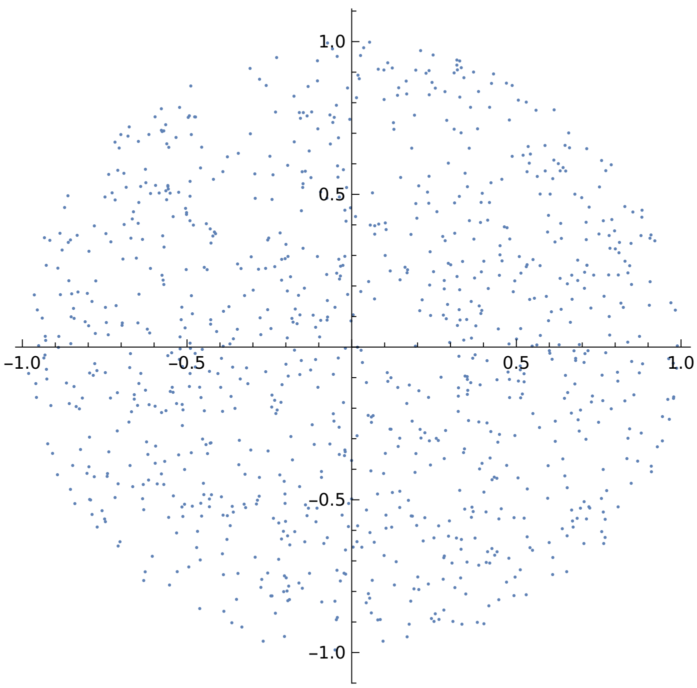
```
dataXY = {}; (*creating empty list to store new points*)
For[i = 1, i ≤ Length[data], i++,
 AppendTo[dataXY, {data[[i, 1]] Cos[data[[i, 2]]], data[[i, 1]] Sin[data[[i, 2]]]}];
 (*polar to cartesian coordinates*)
]
```

Plot of Points Randomly Placed in a Circle

In[•]:=
```
dataXY;
```

In[•]:=
```
ListPlot[dataXY, AspectRatio → 1]
```

Out[•]=



Radial Wave Function of a Hydrogen Atom

Using the Numerov Algorithm to calculate the radial wave function of a hydrogen atom. Follow steps from stark structures paper.

Setting Initial Values

```
In[•]:= g = {0, 0, 0}; (*i-1,i,i+1 positions*)
X = {0, 0, 0};(*i-1,i,i+1 positions*)
i = 2;(*makes code look like equations from notes*)
h = 0.01;(*step size*)
j = 0;
n = 10;
l = 1;

rs = 2 * n (n + 15);(*starting point*)
r2 = rs * e^(-1*h);(*second point*)

x1 = Log[rs];(*to find g[[i-1]]*)
```

$$g[[i-1]] = 2 * e^{2*x1}\left(-e^{-x1} + \frac{1}{2*n^2}\right) + \left(l + \frac{1}{2}\right)^2;$$

```
x2 = Log[r2];(*to find g[[i]]*)
```

$$g[[i]] = 2 * e^{2*x2}\left(-e^{-x2} + \frac{1}{2*n^2}\right) + \left(l + \frac{1}{2}\right)^2;$$

$$X[[i-1]] = 10^{-10};$$
$$X[[i]] = 10^{-5};$$

```
data = {};
r = rs;
```

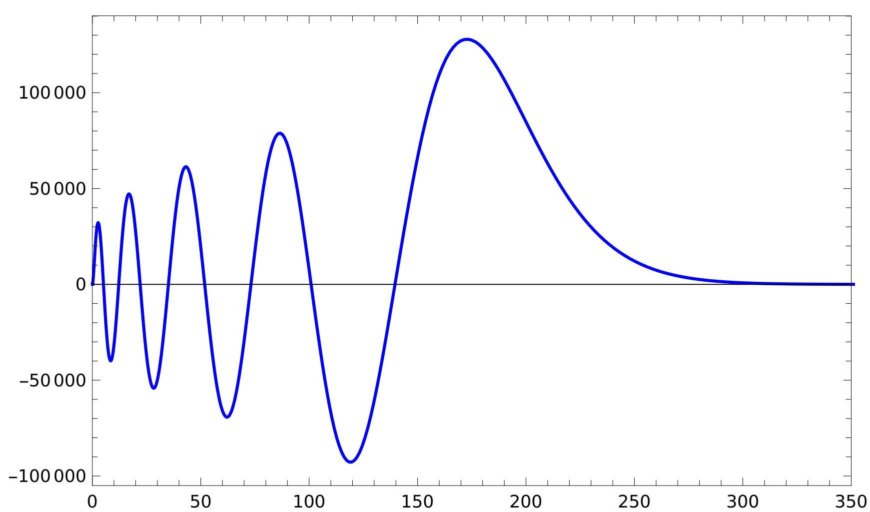
Numerov Algorithm

In[∘]:=
```
For[j = 0, r ≥ 0.05, j++, (*starting at the edge and working way to center*)
    r = rs * e^(-j*h);
    x = Log[r];
    g[[i + 1]] = 2 * e^(2*x) (-e^(-x) + 1/(2 * n^2)) + (l + 1/2)^2; (*new element in g*)
    X[[i + 1]] = (X[[i - 1]] * (g[[i - 1]] - 12/h^2) + X[[i]] * (10 * g[[i]] + 24/h^2)) /
   (12/h^2 - g[[i + 1]]); (*new element in x*)
    AppendTo[data, {r, Sqrt[r] * X[[i + 1]]}];
  (*creating list of points {r,R}*)
    (*updating g & x*)
    g[[i - 1]] = g[[i]];
    g[[i]] = g[[i + 1]];
    X[[i - 1]] = X[[i]];
    X[[i]] = X[[i + 1]];
 ];
```

Plot of Wave Function

In[∘]:=
```
a = ListLinePlot[data, PlotRange → {{0, 350}, All}, PlotStyle → Blue, Frame → True]
```

Out[∘]=



Introduction to Dipole-Dipole Interaction

Using dipole-dipole interaction worksheet and completing problems 1-4. Using a given Hamiltonian, find eigenvalues and eigenvectors. Determining time evolution of a given state and graph probability of being in a second given state.

Determining Time Evolution

```
In[•]:= u = 1.0; (*given value*)
h = 1.0;(*given value*)
H = {{0, u}, {u, 0}}; (*setting up Hamiltonian matrix*)
Eigenvalues[H](*finding eigenvalues of H*)
S = Transpose[Normalize /@ Eigenvectors[H]] (*transpose H*)
U[t_] := Transpose[S].{{e^(i u t/h), 0}, {0, e^(-i u t/h)}}.S

Out[•]= {-1., 1.}

Out[•]= {{-0.707107, 0.707107}, {0.707107, 0.707107}}
```

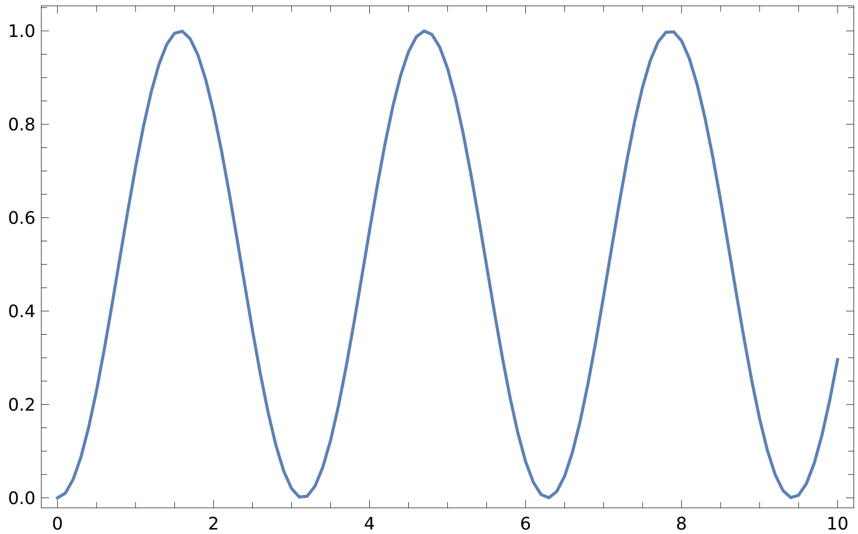
Probability of Being in |p>|p'>State

```
In[•]:= data = {}; (*empty list*)
initial = {{1}, {0}};(*initial conditions, all in s state*)
For[t = 0, t ≤ 10, t = t + 0.1,
  ψ = U[t].initial;
  prob = First[ψ[[2]] * Conjugate[ψ[[2]]]];
  AppendTo[data, {t, prob}];
 ];
```

Graph of Probability

In[•]:=
```
ListPlot[data, Joined → True, Frame → True]
```

Out[•]=



Chapter 2: Dipole-Dipole Experiment

Calibrate Voltage Scan Data

Calibrate voltage scan data and graph experimental data with calculated data. Try to match two graphs as well as possible.

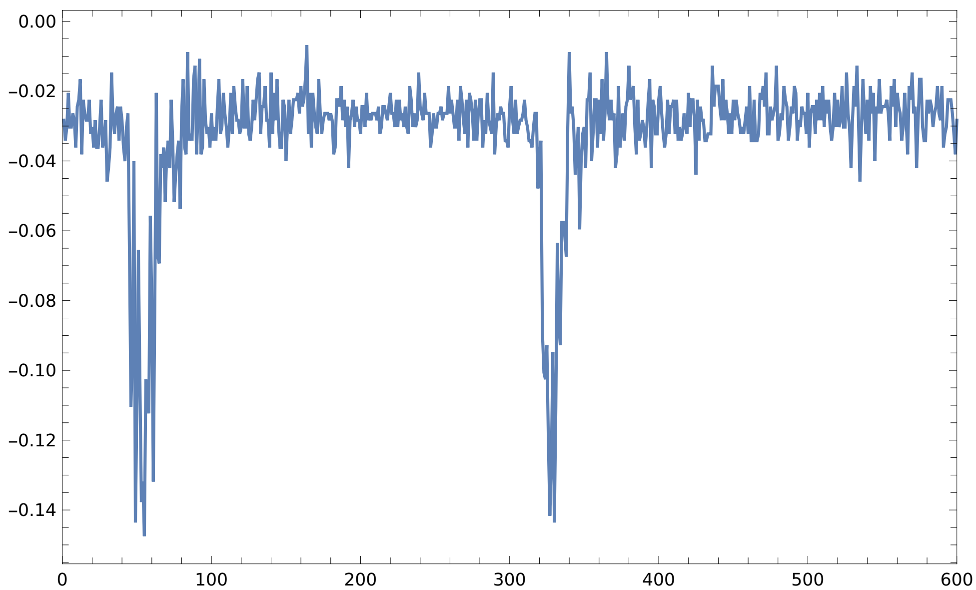
Import Data

```
(* Choose directory and set data file *)
SetDirectory["/home/layoast/Dropbox/research"];
scanset = "2018071111";
(* load metadata file to get points per scan, shots per point, etc *)
mtdStream = OpenRead[scanset <> "_GA_VS.MTD", BinaryFormat → True];
mtd = BinaryReadList[mtdStream, "Integer32"];
Close[mtdStream];
dataStream = OpenRead[scanset <> "_GA_VS.DIP", BinaryFormat → True];
rawData = BinaryReadList[dataStream, "Real32"];
Close[dataStream];
data = Partition[Partition[
    Partition[Partition[rawData, mtd[[1]]], mtd[[5]]], mtd[[2]]], mtd[[3]]][[1]];
```

Graph Data

```
In[∘]:= ListLinePlot[data[[1, 1, 10]], PlotRange → {{0, 600}, All}, Frame → True]
(*data[scan,point,shot]*)
```

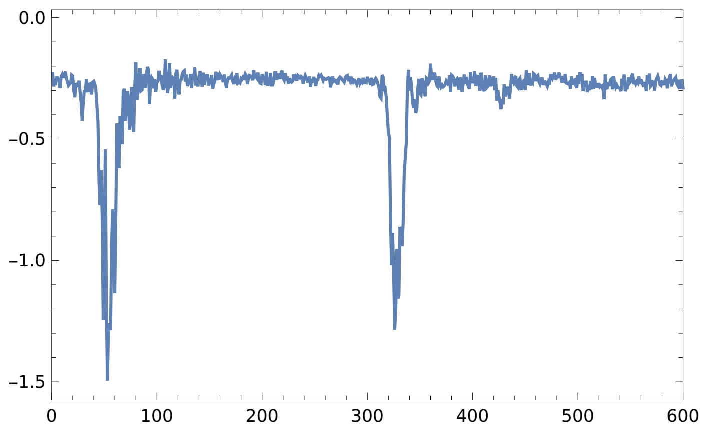
Out[∘]=




```
Dimensions[data]
```

{20, 200, 10, 600}

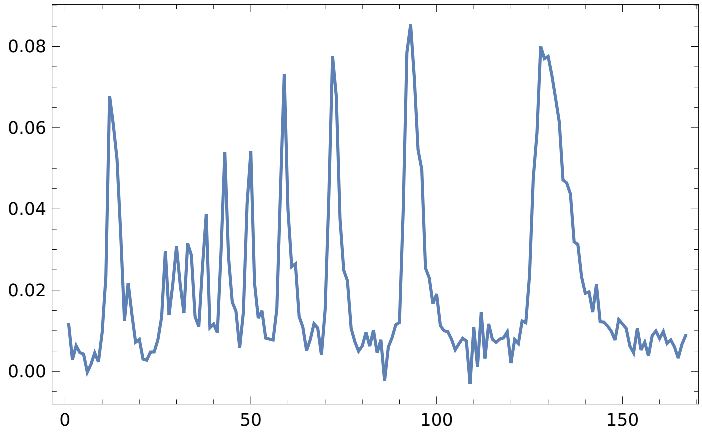
```
ListLinePlot[Sum[data[[1, 1, i]], {i, 1, Length[data[[1, 1]]]}],
 PlotRange → {{0, 600}, All}, Frame → True]
(*data[scan,point,shot]*)
```



Transpose Data and Gate

```
In[•]:= pdata = Transpose[Table[Sum[data[[i, j, k]], {k, 1, Length[data[[i, j]]]}],
     {i, 1, Dimensions[data][[1]]}, {j, 1, Dimensions[data][[2]]}]];
xdata = Table[Sum[pdata[[i, j]], {j, 1, Length[pdata[[i]]]}], {i, 1, Length[pdata]}];
zMin = 500;
zMax = 600;
zero = Sum[-1 * xdata[[i, j]], {j, zMin, zMax}, {i, 1, Length[xdata]}] /
    ((zMax - zMin + 1) * Length[xdata]);
gateMin = 18;
gateMax = 40;
sigGateMin = 18;
sigGateMax = 82;
peakData = Table[Sum[-1 * xdata[[i, j]] - zero, {j, gateMin, gateMax}] / Sum[
      -1 * xdata[[i, j]] - zero, {j, sigGateMin, sigGateMax}], {i, 1, Length[xdata]}];

(*average each set of peaks to find center*)
calData = peakData[[1 ;; 167]] - 0.06;
calData = Table[calData[[i]], {i, Length[calData], 1, -1}];
```

In[•]:=
```
ListLinePlot[calData, PlotRange → All, Frame → True]
```

Out[•]=


In[•]:=
```
(* The specific range of data to be compared to
 the calculations is selected from the above graph. *)
(*gdata =Transpose[ Take[Transpose[zdata],{204,221}]];
gdata2=Sum[Transpose[gdata][[i]],{i,1,Dimensions[gdata][[2]]}];
 gg=Sum[Transpose[gdata][[i]],{i,1,Dimensions[gdata][[2]]}]/Max[gdata2];*)
gg = calData;
(* This function in terms of the calibration and
 field zero shift of the data allows us to adjust these
 parameters for fitting to the calculations later on. *)
Calibrate[calibration_, fieldZeroShift_, vZero_, vScale_] := Module[{},
   vv = Table[i * calibration, {i, 1, 167 + fieldZeroShift}];
  ggvv32 =
    Table[{vv[[i]] + fieldZeroShift, vScale * (gg[[i]] - vZero)}, {i, 1, Length[vv]}]];
```

Graphing Radial Wave Functions

Importing data files including 33s and 33p states and graphing their radial wave functions.

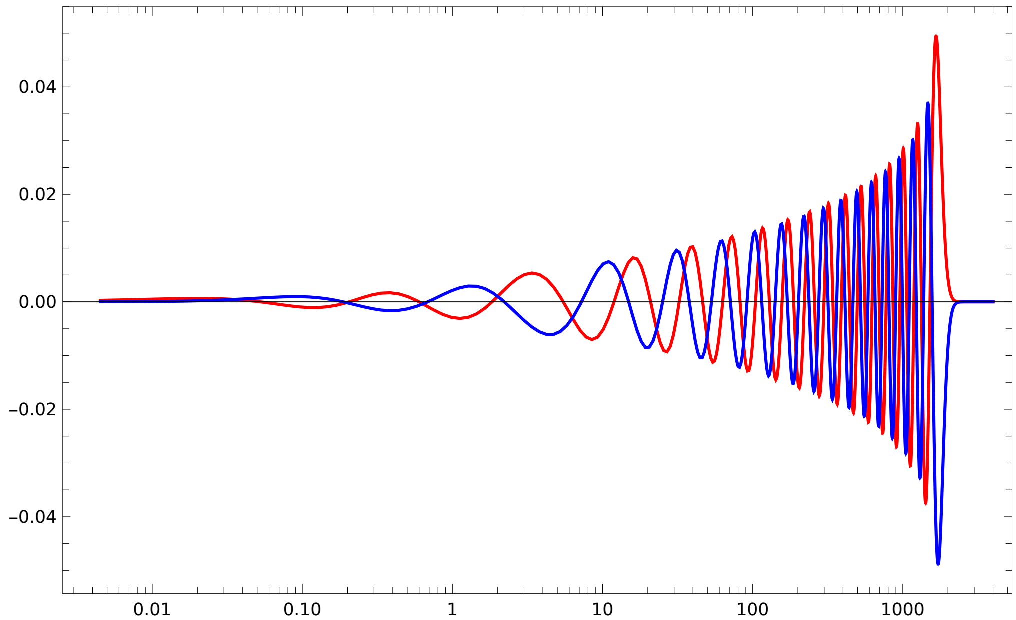
Importing the data files

In[•]:=
```
srad = Import["/home/layoast/Dropbox/research/s33.dat"];
prad = Import["/home/layoast/Dropbox/research/p33.dat"];
```

Graphing 33s and 33p states

In[•]:=
```
ListLogLinearPlot[{{#[[2]], #[[3]]} & /@ srad, {#[[2]], #[[3]]} & /@ prad},
 Joined → True, Frame → True, PlotStyle → {Red, Blue}]
```

Out[•]=



Energy reported by RADIAL for the n=33, l=0 state

$$\left(\frac{(6.626176 * 10^{-34} * 2.9979 * 10^{10})}{-5.604682277398774 * 10^{-4} * 4.3597 * 10^{-18}}\right)^{-1}$$

-123.006

Energy reported by RADIAL for the n=33, l=1 state

$$\left(\frac{(6.626176 * 10^{-34} * 2.9979 * 10^{10})}{-5.427633345386082 * 10^{-4} * 4.3597 * 10^{-18}}\right)^{-1}$$

-119.121

Voltage Scan Analysis

Uses imported data from a voltage scan to calibrate data and graph.

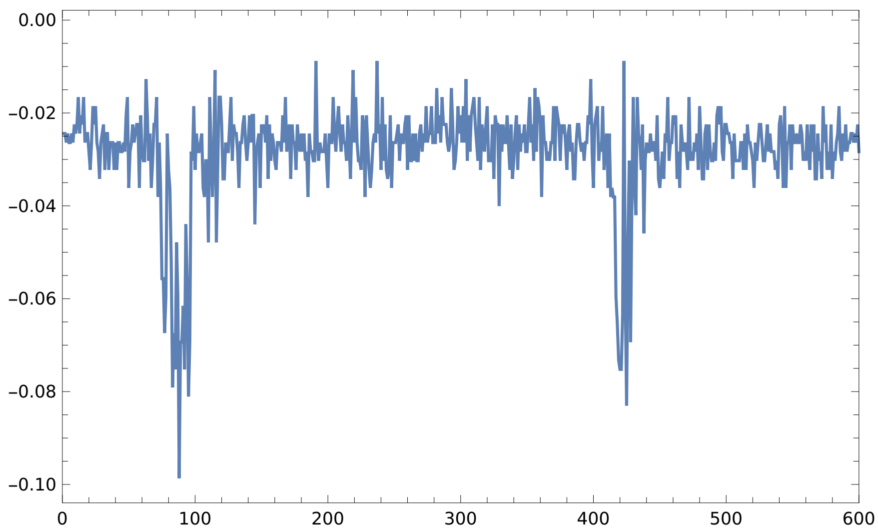
Importing Scan

```
(* Choose directory and set data file *)
SetDirectory["/home/layoast/Dropbox/research"];
scanset = "2018071611";
(* load metadata file to get points per scan, shots per point, etc *)
mtdStream = OpenRead[scanset <> "_GA_VS.MTD", BinaryFormat → True];
mtd = BinaryReadList[mtdStream, "Integer32"];
Close[mtdStream];
dataStream = OpenRead[scanset <> "_GA_VS.DIP", BinaryFormat → True];
rawData = BinaryReadList[dataStream, "Real32"];
Close[dataStream];
data = Partition[Partition[
    Partition[Partition[rawData, mtd[[1]]], mtd[[5]]], mtd[[2]]], mtd[[3]]][[1]];
```

Plotting Partitioned Data

```
In[•]:= ListLinePlot[data[[1, 1, 10]], PlotRange → {{0, 600}, All}, Frame → True]
(*data[scan,point,shot]*)
```

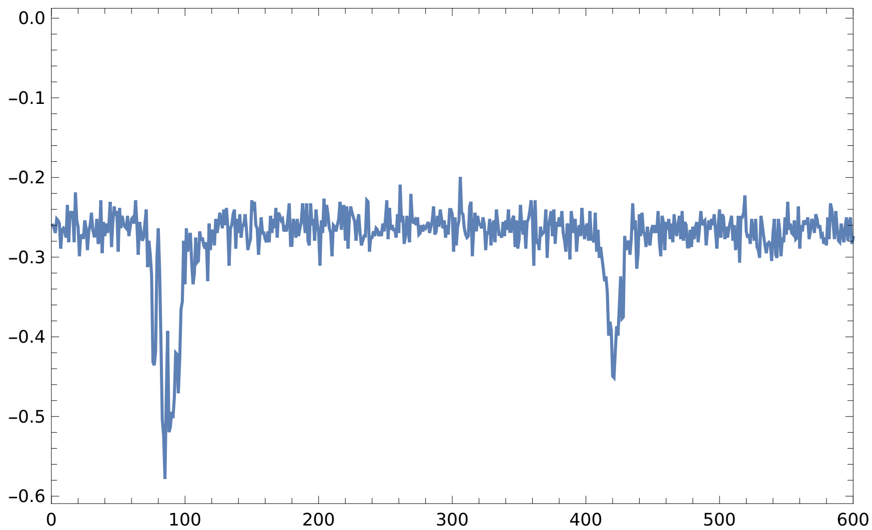
Out[•]=




```
In[•]:= Dimensions[data]
Out[•]= {10, 267, 10, 600}
```

In[∘]:=
```
ListLinePlot[Sum[data[[1, 1, i]], {i, 1, Length[data[[1, 1]]]}],
 PlotRange → {{0, 600}, All}, Frame → True]
(*data[scan,point,shot]*)
```

Out[∘]=



Transposing Data and Selecting a Gate

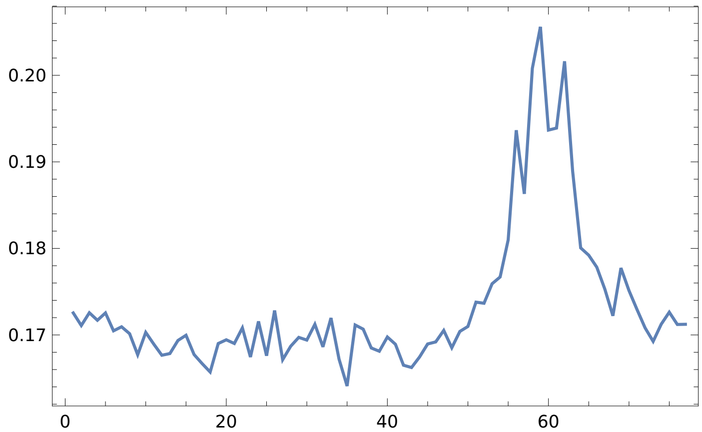
In[∘]:=
```
pdata = Transpose[Table[Sum[data[[i, j, k]], {k, 1, Length[data[[i, j]]]}],
    {i, 1, Dimensions[data][[1]]}, {j, 1, Dimensions[data][[2]]}]];
xdata = Table[Sum[pdata[[i, j]], {j, 1, Length[pdata[[i]]]}], {i, 1, Length[pdata]}];
```

In[∘]:=
```
zMin = 500;
zMax = 600;
zero = Sum[-1 * xdata[[i, j]], {j, zMin, zMax}, {i, 1, Length[xdata]}] /
   ((zMax - zMin + 1) * Length[xdata]);
```

Plotting Transposed and Calibrated Data

```
(*average each set of peaks to find center*)
calData = peakData[[124 ;;]];
calData = Table[calData[[i]], {i, 1, Length[calData], 1}];
ListLinePlot[calData, PlotRange → All, Frame → True]
```

Out[∘]=

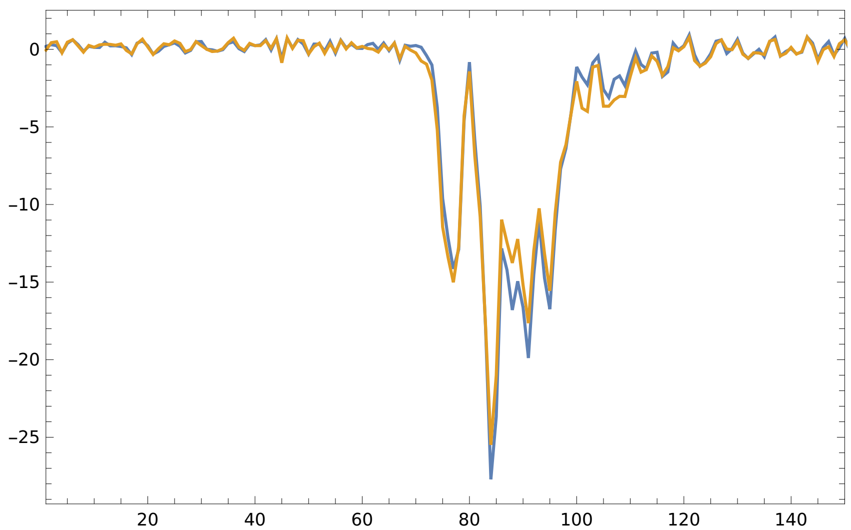


In[∘]:= `Dimensions[xdata]`

Out[∘]= {267, 600}

In[∘]:=
```
ListLinePlot[{xdata[[9]] + zero, xdata[[68]] + zero},
 PlotRange → {{1, 150}, All}, Frame → True]
```

Out[∘]=



Genetic Algorithm Analysis

Imports data from a GA scan and graphs. Before and after of GA can be seen.

Import Data

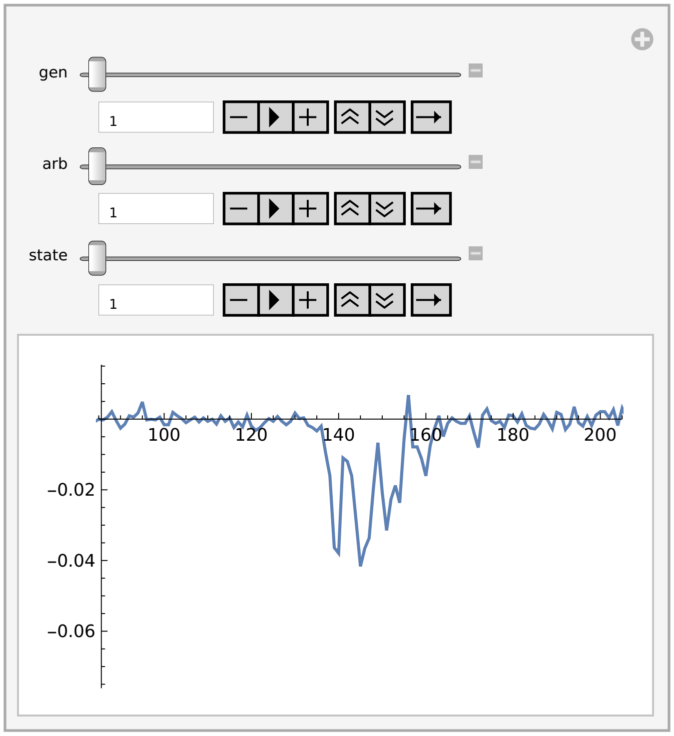
In[179]:=
```
(* BE SURE TO CHANGE "scanset" VARIABLE TO MATCH FILENAME OF INTEREST *)
(* ONLY USE WITH GENETIC ALGORITHM SCANS FROM 1/25/17 OR LATER *)
SetDirectory["/home/layoast/Dropbox/research"];
scanset = "2018071210"
metafilestream = OpenRead[scanset <> "_GA_A.MTD", BinaryFormat → True];
(*reading one item at a time*)
TextCell[Row[{pointsPerTrace = BinaryRead[metafilestream, "Integer32"],
   " points per trace"}]]
popSize = BinaryRead[metafilestream, "Integer32"];
TextCell[Row[{numGenerations  = BinaryRead[metafilestream, "Integer32"],
   " generations completed"}]]
(*TextCell[Row[{sets  = BinaryRead[metafilestream,"Integer32"]," set(s)"}]]*)
BinaryRead[metafilestream, "Integer32"];
(* this parameter is currently meaningless *)
TextCell[
 Row[{"mutation rate = ", mutationRate = BinaryRead[metafilestream, "Real64"]}]]
TextCell[Row[{"tournament size = ",
   tournamentSize = BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"elitisim number = ",
   elitismNumber = BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"gene size = ", geneSize =
    BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"target gate =  ", minGate = BinaryRead[metafilestream, "Real64"],
   " - ", maxGate = BinaryRead[metafilestream, "Real64"]}]]
minGate2 = BinaryRead[metafilestream, "Real64"];
maxGate2 = BinaryRead[metafilestream, "Real64"];
TextCell[Row[
  {"total signal gate =  ", minTotalGate = BinaryRead[metafilestream, "Integer32"],
   " - ", maxTotalGate = BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"zero level gate =  ",
   minZeroGate = BinaryRead[metafilestream, "Integer32"], " - ",
   maxZeroGate = BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"total shots per arb = ",
   shotsPerArb = BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"consecutive shots = ",
   consecutiveShots = BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"length of arb (ns) = ",
```

```
    pulseLength = BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"fitness type = ", fitnessType =
     BinaryRead[metafilestream, "Integer32"]}]]
TextCell[Row[{"total traces = ", totalTraces =
     numGenerations * popSize * shotsPerArb}]]
If[fitnessType == 7 || fitnessType == 8 || fitnessType == 10 ||
    fitnessType == 11 || fitnessType == 12, twoState = 2, twoState = 1];
TextCell[Row[{"number of states = ", twoState}]]
popSize /= twoState;
TextCell[Row[{"population size = ", popSize}]]
Close[metafilestream];
avgTraces = N@Partition[Import[scanset <> ".AVG", "Real64"], pointsPerTrace];
avgTraces = Partition[avgTraces, twoState];
avgTraces = Partition[avgTraces, popSize];
fitnessScores = N@Partition[Import[scanset <> ".FIT", "Real64"], popSize];
AvgFitnessScore = Table[
    Mean[Table[fitnessScores[[i, j]], {j, 1, Length[fitnessScores[[i]]] - 1, 1}]],
    {i, 1, Length[fitnessScores], 1}];
MinFitnessScore = Table[Min[Table[fitnessScores[[i, j]],
      {j, 1, Length[fitnessScores[[i]]] - 1, 1}]], {i, 1, Length[fitnessScores], 1}];
MaxFitnessScore = Table[Max[Table[fitnessScores[[i, j]],
      {j, 1, Length[fitnessScores[[i]]] - 1, 1}]], {i, 1, Length[fitnessScores], 1}];
TextCell[Row[{"Dimensions[avgTraces] = ", Dimensions[avgTraces]}   ]]
TextCell[Row[{"Dimensions[fitnessScores] = ", Dimensions[fitnessScores]}   ]]
Manipulate[
  ListPlot[avgTraces[[gen, arb, state]], Joined → True, ImageSize → 300, PlotRange →
    {{minTotalGate - 25, maxTotalGate + 25}, {Min[avgTraces], Max[avgTraces]}}],
  {gen, 1, numGenerations, 1, Appearance → "Open"},
  {arb, 1, popSize, 1, Appearance → "Open"},
  {state, 1, twoState, 1, Appearance → "Open"}]
(*Dynamic[Manipulate[ListPlot[{AvgFitnessScore,MaxFitnessScore,MinFitnessScore},
    Joined→True,PlotRange->{{0,numGenerations},{0,1}},
    ImageSize→1000,GridLines→{{LastGoodGen},{}}],
   {LastGoodGen,0,numGenerations,1,Appearance→"Open"}]]*)
LastGoodGen = 1;
fitnessRange = Max[MaxFitnessScore] - Min[MinFitnessScore];
{Dynamic[ListPlot[{AvgFitnessScore, MaxFitnessScore, MinFitnessScore},
    Joined → True, PlotRange -> {{0, numGenerations}, {0, 1}},
    ImageSize → 300, AspectRatio → 1, GridLines → {{LastGoodGen}, {}}]],
  Dynamic[ListPlot[{AvgFitnessScore, MaxFitnessScore, MinFitnessScore},
    Joined → True, PlotRange -> {{0, numGenerations}, {Min[MinFitnessScore] -
         fitnessRange * 0.1, Max[MaxFitnessScore] + fitnessRange * 0.1}},
    AxesOrigin → {0, Min[MinFitnessScore] - fitnessRange * 0.1}, ImageSize → 300,
    AspectRatio → 1, GridLines → {{LastGoodGen}, {}}]]}
```

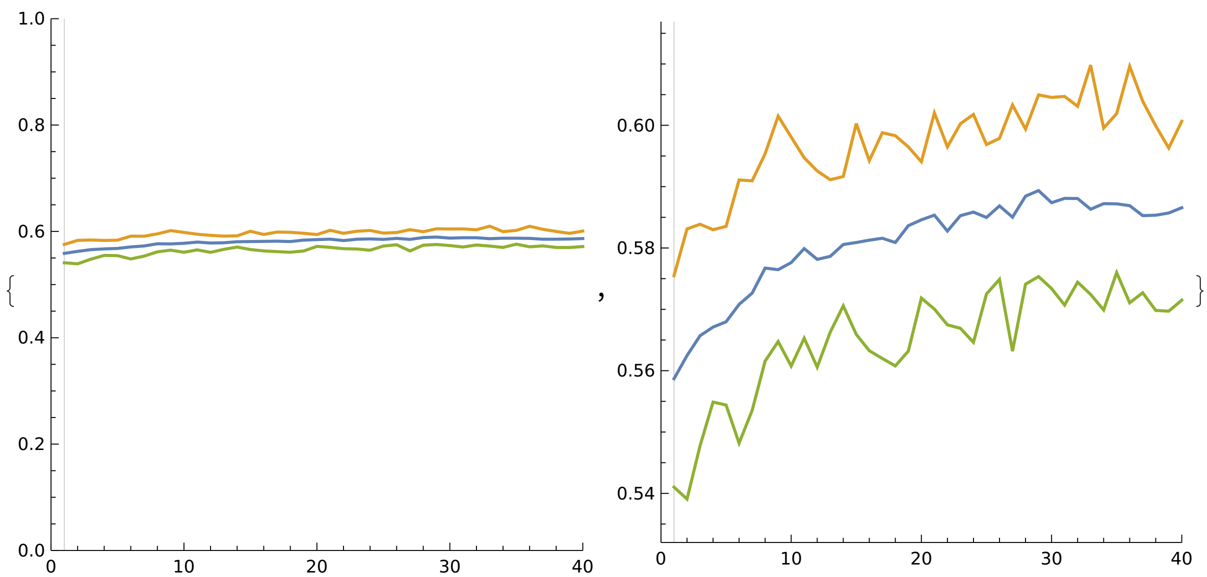
```
TextCell[Row[{"LastGoodGen:  ", Manipulator[Dynamic[LastGoodGen],
    {1, numGenerations, 1}, Appearance → "Open"]}   ]]
```

```
Out[180]= 2018071210
Out[182]= 600 points per trace
Out[184]= 40 generations completed
Out[186]= mutation rate = 0.005
Out[187]= tournament size = 4
Out[188]= elitisim number = 8
Out[189]= gene size = 1
Out[190]= target gate =  36. - 49.
Out[193]= total signal gate =  110 - 180
Out[194]= zero level gate =  499 - 599
Out[195]= total shots per arb = 10
Out[196]= consecutive shots = 10
Out[197]= length of arb (ns) = 0
Out[198]= fitness type = 10
Out[199]= total traces = 40 000
Out[201]= number of states = 2
Out[203]= population size = 50
Out[212]= Dimensions[avgTraces] = {40, 50, 2, 600}
Out[213]= Dimensions[fitnessScores] = {40, 50}
```

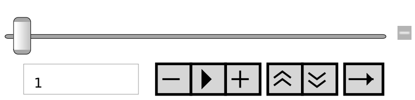
Out[214]=



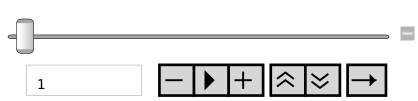
Out[217]=



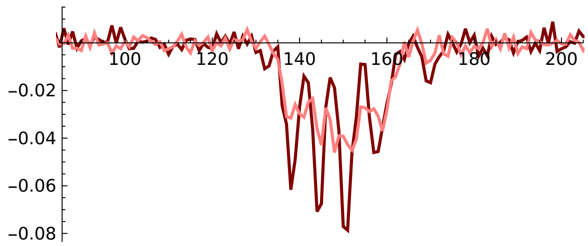
Out[218]= `LastGoodGen:`



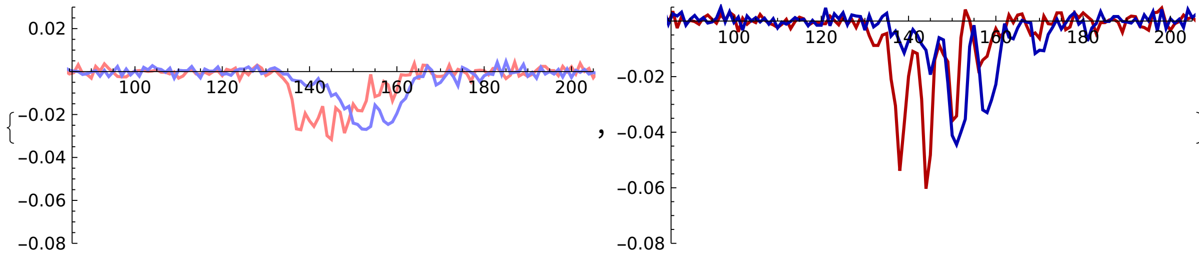
```
In[219]:= maxFitnessIndex =
   Position[fitnessScores[[LastGoodGen]], MaxFitnessScore[[LastGoodGen]]][[1, 1]];
  TextCell[Row[{"LastGoodGen:  ", Manipulator[Dynamic[LastGoodGen],
       {1, numGenerations, 1}, Appearance → "Open"]}   ]]
  Dynamic[ListPlot[{avgTraces[[LastGoodGen, maxFitnessIndex, 1]] +
       avgTraces[[LastGoodGen, maxFitnessIndex, 2]],
      avgTraces[[1, popSize, 1]] + avgTraces[[1, popSize, 2]]},
     Joined → True, ImageSize → 300, AspectRatio → 0.45,
     PlotRange → {{minTotalGate - 25, maxTotalGate + 25}, All},
     PlotStyle → {Darker[Red, 0.5], Lighter[Red, 0.5]}]]
  Dynamic[{ListPlot[{avgTraces[[1, popSize, 1]], avgTraces[[1, popSize, 2]]},
      Joined → True, ImageSize → 300, AspectRatio → 0.45,
      PlotRange → {{minTotalGate - 25, maxTotalGate + 25}, {-0.08, 0.03}},
      PlotStyle → {Lighter[Red, 0.5], Lighter[Blue, 0.5]}],
     ListPlot[{avgTraces[[LastGoodGen, maxFitnessIndex, 1]],
       avgTraces[[LastGoodGen, maxFitnessIndex, 2]]}, Joined → True, ImageSize → 300,
      AspectRatio → 0.45, PlotRange → {{minTotalGate - 25, maxTotalGate + 25},
        (*{Min[avgTraces],Max[avgTraces]}*){-0.08, 0.005}},
      PlotStyle → {Darker[Red, 0.3], Darker[Blue, 0.3]}]}]
  Dynamic[{ListPlot[{avgTraces[[LastGoodGen, maxFitnessIndex, 1]],
       avgTraces[[1, popSize, 1]]}, Joined → True, ImageSize → 300,
      AspectRatio → 0.45, PlotRange → {{minTotalGate - 25, maxTotalGate + 25}, All},
      PlotStyle → {Darker[Red, 0.5], Lighter[Red, 0.5]}], ListPlot[
      {avgTraces[[LastGoodGen, maxFitnessIndex, 2]], avgTraces[[1, popSize, 2]]},
      Joined → True, ImageSize → 300, AspectRatio → 0.45,
      PlotRange → {{minTotalGate - 25, maxTotalGate + 25}, All},
      PlotStyle → {Darker[Blue, 0.5], Lighter[Blue, 0.5]}]}]

  (*Dynamic[ListPlot[{AvgFitnessScore,MaxFitnessScore,MinFitnessScore},
     Joined→True,PlotRange->{{0,LastGoodGen},{0.49,0.62}},
     AxesOrigin→{0,0.49},ImageSize→400,AspectRatio→1]]*)
  Dynamic[ListPlot[{AvgFitnessScore, MaxFitnessScore, MinFitnessScore}, Joined → True,
     PlotRange -> {{0, LastGoodGen}, {Min[MinFitnessScore] - fitnessRange * 0.1,
         Max[MaxFitnessScore] + fitnessRange * 0.1}},
     AxesOrigin → {0, Min[MinFitnessScore] - fitnessRange * 0.1},
     ImageSize → 300, AspectRatio → 1]]

Out[220]= LastGoodGen:
```



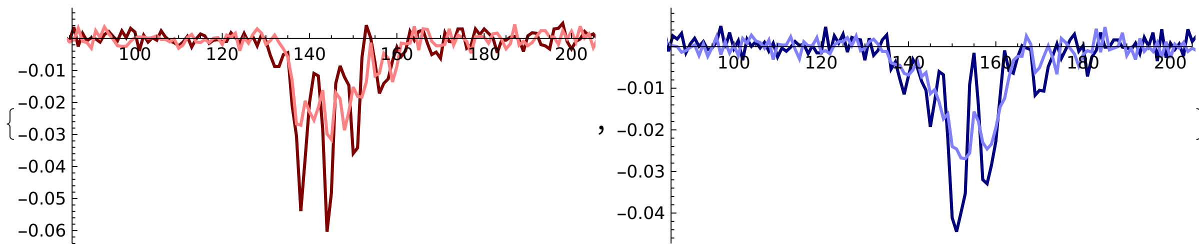
Out[221]=



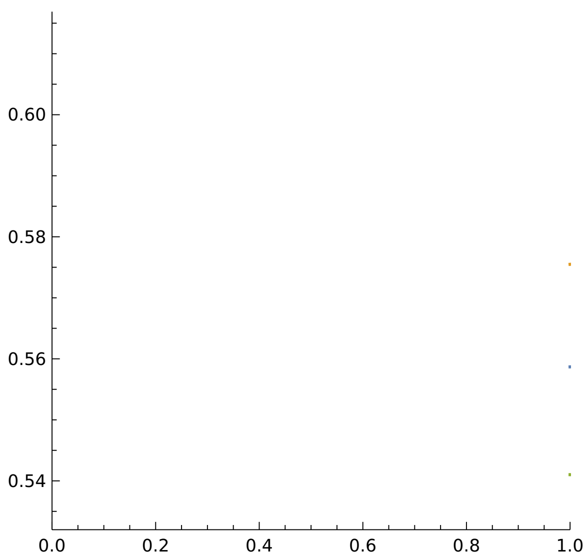
Out[222]=



Out[223]=

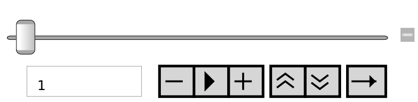


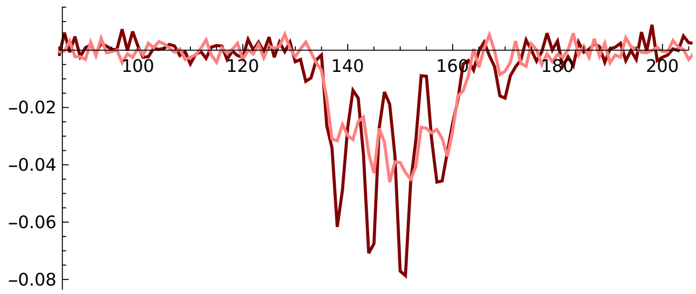
Out[224]=

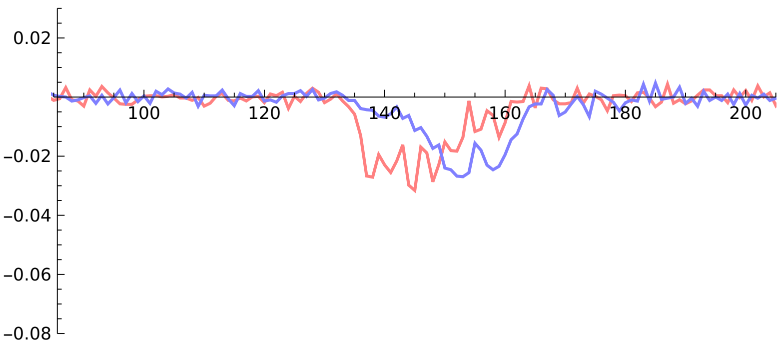
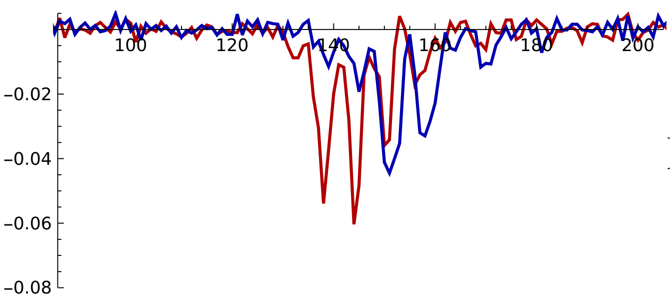


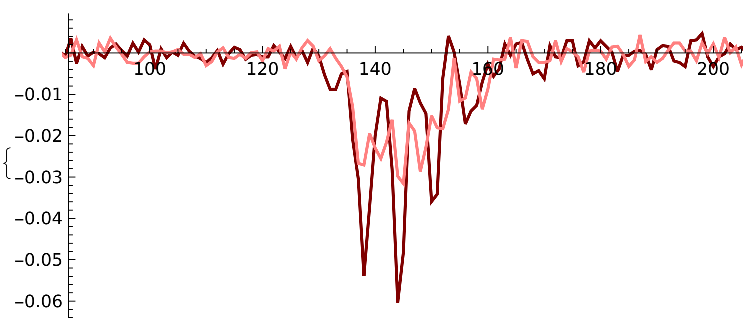
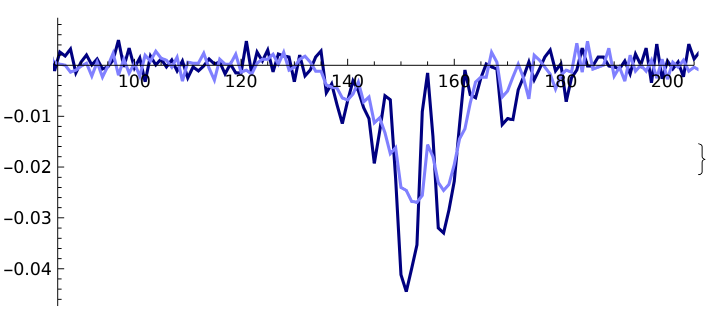
Use Animation to Show Progress of GA from First to Last Generation

Out[∘]= LastGoodGen:



Out[•]= 
100
120
140
160
180
200
-0.02
-0.04
-0.06
-0.08

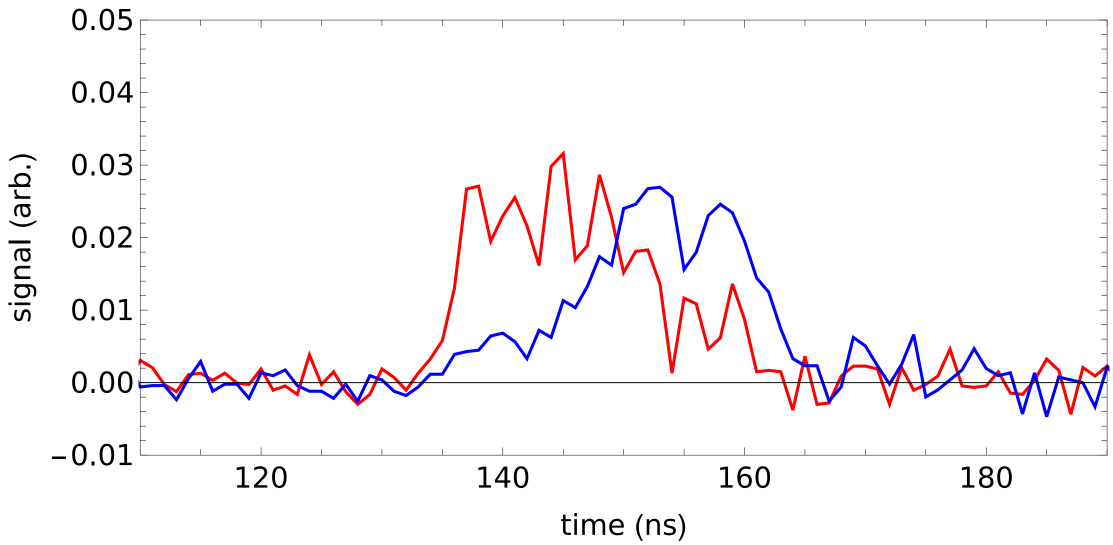
Out[•]= { 
0.02
100
120
140
160
180
200
-0.02
-0.04
-0.06
-0.08
,

100
120
140
160
180
200
-0.02
-0.04
-0.06
-0.08
}

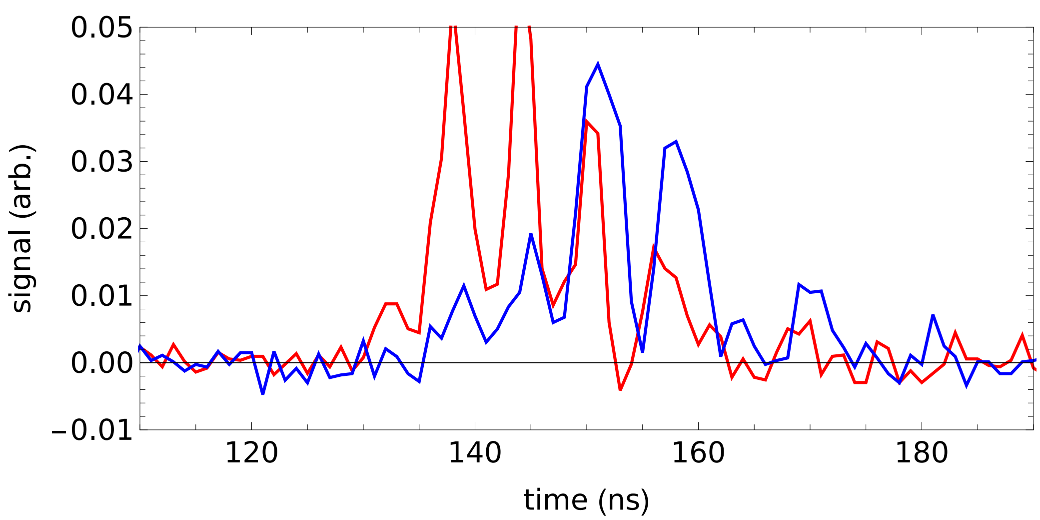
Out[•]= { 
100
120
140
160
180
200
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
,

100
120
140
160
180
200
-0.01
-0.02
-0.03
-0.04
}

Before and After of GA

In[•]:=
```
a = (ListPlot[{-1 * avgTraces[[1, popSize, 1]], -1 * avgTraces[[1, popSize, 2]]},
    Joined → True, ImageSize → 800, AspectRatio → 0.45,
    PlotRange → {{110, 190}, {-0.01, 0.05}}, PlotStyle → {Red, Blue, 0.5},
    Frame → True, FrameLabel → {"time (ns)", "signal (arb.)"},
    BaseStyle → {FontSize → 15, FontColor → Black}, ImageSize → 300] ×
  ListPlot[{-1 * avgTraces[[LastGoodGen, maxFitnessIndex, 1]],
     -1 * avgTraces[[LastGoodGen, maxFitnessIndex, 2]]}, Joined → True,
    FrameLabel → {"time (ns)", "signal (arb.)"}, ImageSize → 800, AspectRatio → 0.45,
    PlotRange → {{110, 190}, (*{Min[avgTraces],Max[avgTraces]}*){-0.01, 0.05}},
    PlotStyle → {Red, Blue}, Frame → True,
    BaseStyle → {FontSize → 15, FontColor → Black}, ImageSize → 300])
```

Out[•]=





In[•]:=
```
Export["/home/layoast/Dropbox/research/gaBeforeAfter.eps", a]
```
Out[•]=
```
/home/layoast/Dropbox/research/gaBeforeAfter.eps
```

```
TextCell[
 Row[{"Overlap, taking every point into account (whether negative or not)"}]]
InitialOverlap = Sum[avgTraces[[1, popSize, 1, i]] * avgTraces[[1, popSize, 2, i]],
    {i, minTotalGate, maxTotalGate, 1}] /
   (Sum[avgTraces[[1, popSize, 1, i]] * avgTraces[[1, popSize, 1, i]],
       {i, minTotalGate, maxTotalGate, 1}] * Sum[avgTraces[[1, popSize, 2, i]] *
        avgTraces[[1, popSize, 2, i]], {i, minTotalGate, maxTotalGate, 1}])^1/2
FinalOverlap = Sum[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] * avgTraces[[
      LastGoodGen, maxFitnessIndex, 2, i]], {i, minTotalGate, maxTotalGate, 1}] /
   (Sum[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] * avgTraces[[LastGoodGen,
         maxFitnessIndex, 1, i]], {i, minTotalGate, maxTotalGate, 1}] *
      Sum[avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]] * avgTraces[[LastGoodGen,
         maxFitnessIndex, 2, i]], {i, minTotalGate, maxTotalGate, 1}])^1/2
(*InitialOverlap=Sum[If[avgTraces[[1,popSize,1,i]]>0&&
       avgTraces[[1,popSize,2,i]]>0,avgTraces[[1,popSize,1,i]]*
       avgTraces[[1,popSize,2,i]],0],{i,minTotalGate,maxTotalGate,1}]
    FinalOverlap=Sum[If[avgTraces[[LastGoodGen,maxFitnessIndex,1,i]]>0&&
       avgTraces[[LastGoodGen,maxFitnessIndex,2,i]]>0,
      avgTraces[[LastGoodGen,maxFitnessIndex,1,i]]*avgTraces[[LastGoodGen,
        maxFitnessIndex,2,i]],0],{i,minTotalGate,maxTotalGate,1}]
    FinalOverlap/InitialOverlap
    fitnessScores[[1,popSize]]
    fitnessScores[[LastGoodGen,maxFitnessIndex]]*)

TextCell[Row[{"Overlap, if ignoring negative signal"}]]
InitialOverlap =
 Sum[If[avgTraces[[1, popSize, 1, i]] > 0, 0, avgTraces[[1, popSize, 1, i]]] *
     If[avgTraces[[1, popSize, 2, i]] > 0, 0, avgTraces[[1, popSize, 2, i]]],
    {i, minTotalGate, maxTotalGate, 1}] /
   (Sum[If[avgTraces[[1, popSize, 1, i]] > 0, 0, avgTraces[[1, popSize, 1, i]]]^2,
       {i, minTotalGate, maxTotalGate, 1}] *
      Sum[If[avgTraces[[1, popSize, 2, i]] > 0, 0, avgTraces[[1, popSize, 2, i]]]^2,
       {i, minTotalGate, maxTotalGate, 1}])^1/2
FinalOverlap = Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] > 0,
      0, avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]]] *
     If[avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]] > 0, 0, avgTraces[[
        LastGoodGen, maxFitnessIndex, 2, i]]], {i, minTotalGate, maxTotalGate, 1}] /
   (Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] > 0, 0, avgTraces[[
          LastGoodGen, maxFitnessIndex, 1, i]]]^2, {i, minTotalGate, maxTotalGate,
```

```
1}] * Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]] > 0,
  0, avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]]]^2,
 {i, minTotalGate, maxTotalGate, 1}])^(1/2)
```

```
TextCell[Row[
  {"Overlap, only throwing away points where BOTH traces are negative (ignoring
     double-negatives)"}]]
InitialOverlap = Sum[If[avgTraces[[1, popSize, 1, i]] > 0 &&
     avgTraces[[1, popSize, 2, i]] > 0, 0, avgTraces[[1, popSize, 1, i]] *
     avgTraces[[1, popSize, 2, i]]], {i, minTotalGate, maxTotalGate, 1}] /
  (Sum[If[avgTraces[[1, popSize, 1, i]] > 0 && avgTraces[[1, popSize, 2, i]] > 0,
       0, avgTraces[[1, popSize, 1, i]]]^2, {i, minTotalGate, maxTotalGate, 1}] *
     Sum[If[avgTraces[[1, popSize, 1, i]] > 0 && avgTraces[[1, popSize, 2, i]] > 0,
       0, avgTraces[[1, popSize, 2, i]]]^2, {i, minTotalGate, maxTotalGate, 1}])^(1/2)
FinalOverlap = Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] > 0 &&
     avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]] > 0, 0,
    avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] * avgTraces[[LastGoodGen,
      maxFitnessIndex, 2, i]]], {i, minTotalGate, maxTotalGate, 1}] /
  (Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] > 0 && avgTraces[[
          LastGoodGen, maxFitnessIndex, 2, i]] > 0, 0, avgTraces[[LastGoodGen,
        maxFitnessIndex, 1, i]]]^2, {i, minTotalGate, maxTotalGate, 1}] *
     Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] > 0 &&
        avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]] > 0,
       0, avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]]]^2,
      {i, minTotalGate, maxTotalGate, 1}])^(1/2)
```

Overlap, taking every point into account (whether negative or not)

0.614171

0.549357

Overlap, if ignoring negative signal

0.6175

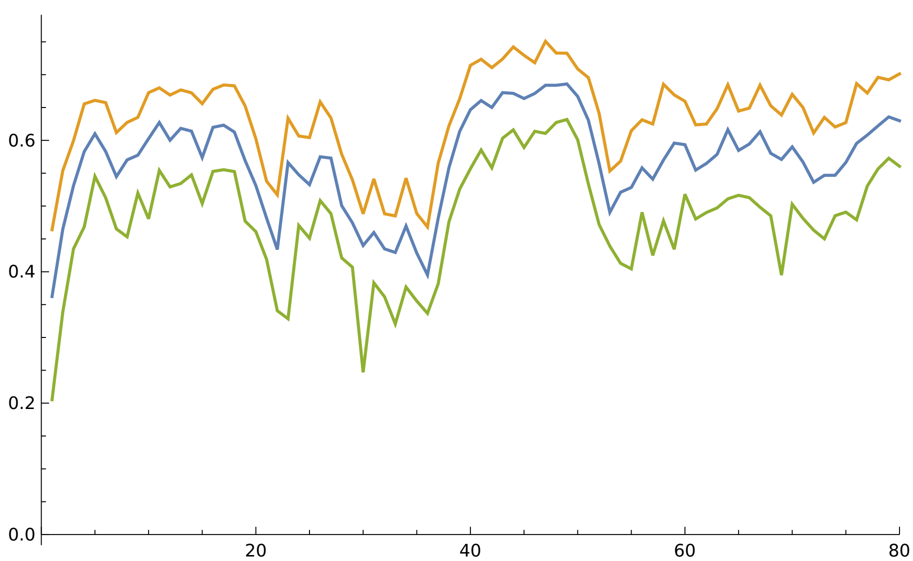
0.5545

Overlap, only throwing away points where BOTH traces are negative (ignoring double-negatives)

0.614024

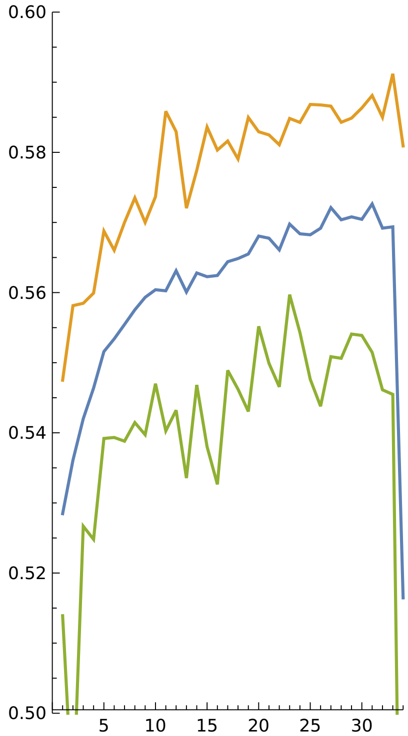
0.549343

```
LogisticDecay[x_, k_, x1_, x2_] :=
  ((Exp[- (k / 10 000) * x1] - Exp[- (k / 10 000) * (-x + (x2 + x1))]) /
     (Exp[- (k / 10 000) * x1] - Exp[- (k / 10 000) * x2])) *
   ((1 + Exp[- (k / 10 000) * x2]) / (1 + Exp[- (k / 10 000) * (-x + (x2 + x1))]));
LogisticGrowth[x_, k_, x1_, x2_] :=
  ((Exp[- (k / 10 000) * x1] - Exp[- (k / 10 000) * x]) / (Exp[- (k / 10 000) * x1] - Exp[- (k / 10 000) * x2])) * ((1 + Exp[- (k / 10 000) * x2]) / (1 + Exp[- (k / 10 000) * x]));
NewFitness = Table[Table[
     (Sum[If[avgTraces[[i, j, 1, k]] < 0, avgTraces[[i, j, 1, k]], 0] * LogisticDecay[k,
           1, minTotalGate, maxTotalGate], {k, minTotalGate, maxTotalGate, 1}] *
        Sum[If[avgTraces[[i, j, 2, k]] < 0, avgTraces[[i, j, 2, k]], 0] *
          LogisticGrowth[k, 1, minTotalGate, maxTotalGate],
         {k, minTotalGate, maxTotalGate, 1}])^(1/2)
     , {j, 1, popSize, 1}], {i, 1, numGenerations, 1}];
NewAvgFitnessScore =
  Table[Mean[Table[NewFitness[[i, j]], {j, 1, Length[NewFitness[[i]]] - 1, 1}]],
   {i, 1, Length[NewFitness], 1}];
NewMinFitnessScore = Table[Min[Table[NewFitness[[i, j]],
     {j, 1, Length[NewFitness[[i]]] - 1, 1}]], {i, 1, Length[NewFitness], 1}];
NewMaxFitnessScore = Table[Max[Table[NewFitness[[i, j]],
     {j, 1, Length[NewFitness[[i]]] - 1, 1}]], {i, 1, Length[NewFitness], 1}];
ListPlot[{NewAvgFitnessScore, NewMaxFitnessScore, NewMinFitnessScore},
 Joined → True, PlotRange -> {{0, numGenerations}, All}, ImageSize → 1000]
```



```
unperturbed = Table[Table[-avgTraces[[i, popSize, k]],
     {k, minTotalGate, maxTotalGate, 1}], {i, 1, numGenerations, 1}];
signalLevel = Table[Sum[unperturbed[[i, j]], {j, 1, Length[unperturbed[[i]]], 1}],
    {i, 1, numGenerations, 1}];
signalLevel = Table[signalLevel[[i]] / Max[signalLevel],
    {i, 1, Length[signalLevel], 1}];
Manipulate[ListPlot[unperturbed[[i]], Joined → True, PlotRange → {0, 0.5}],
  {i, 1, Length[unperturbed], 1, Appearance → "Open"}]
TwoAxisListLinePlot[{f_, g_}] := Module[
   {fgraph, ggraph, frange, grange, fticks, gticks}, {fgraph, ggraph} = MapIndexed[
     ListLinePlot[#, Axes → True, PlotStyle → ColorData[1][#2[[1]]]] &, {f, g}];
   {frange, grange} = Last[PlotRange /. AbsoluteOptions[#, PlotRange]] & /@
     {fgraph, ggraph};
   fticks = Last[Ticks /. AbsoluteOptions[fgraph, Ticks]] /.
     _RGBColor | _GrayLevel | _Hue :→ ColorData[1][1];
   gticks = (MapAt[Function[r, Rescale[r, grange, frange]], #, {1}] & /@
       Last[Ticks /. AbsoluteOptions[ggraph, Ticks]]) /.
     _RGBColor | _GrayLevel | _Hue → ColorData[1][2];
   Show[fgraph, ggraph /. Graphics[graph_, s___] :→
      Graphics[GeometricTransformation[graph,
        RescalingTransform[{{0, 1}, grange}, {{0, 1}, frange}]], s], Axes → False,
    Frame → True, FrameStyle → {ColorData[1] /@ {1, 2}, {Automatic, Transparent}},
    FrameTicks → {{fticks, gticks}, {Automatic, Automatic}}]]
{ListPlot[{signalLevel, AvgFitnessScore}, Joined → True],
 TwoAxisListLinePlot[{AvgFitnessScore, signalLevel}]}
```

```
ListPlot[{AvgFitnessScore, MaxFitnessScore, MinFitnessScore}, Joined → True,
 PlotRange -> {{0, numGenerations}, {0.5, 0.6}}, ImageSize → 300, AspectRatio → 2]
```




```
InitialSig = Sum[avgTraces[[1, popSize, 2, i]] * avgTraces[[1, popSize, 2, i]],
  {i, 1, Length[avgTraces[[1, popSize, 1]]], 1}]
FinalSig = Sum[avgTraces[[numGenerations, popSize, 2, i]] *
   avgTraces[[numGenerations, popSize, 2, i]],
  {i, 1, Length[avgTraces[[numGenerations, popSize, 1]]], 1}]
PercentSig = (FinalSig - InitialSig) / InitialSig
```

```
0.0804757

0.0971984

0.207799
```

```
(* overlap, if ignoring negative signal *)
InitialOverlap =
 Sum[If[avgTraces[[1, popSize, 1, i]] > 0, 0, avgTraces[[1, popSize, 1, i]]] *
    If[avgTraces[[1, popSize, 2, i]] > 0, 0, avgTraces[[1, popSize, 2, i]]],
   {i, minTotalGate, maxTotalGate, 1}] /
  (Sum[If[avgTraces[[1, popSize, 1, i]] > 0, 0, avgTraces[[1, popSize, 1, i]]]^2,
      {i, minTotalGate, maxTotalGate, 1}] *
     Sum[If[avgTraces[[1, popSize, 2, i]] > 0, 0, avgTraces[[1, popSize, 2, i]]]^2,
      {i, minTotalGate, maxTotalGate, 1}])^(1/2)
FinalOverlap = Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] > 0,
      0, avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]]] *
    If[avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]] > 0, 0, avgTraces[[
       LastGoodGen, maxFitnessIndex, 2, i]]], {i, minTotalGate, maxTotalGate, 1}] /
  (Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 1, i]] > 0, 0, avgTraces[[
           LastGoodGen, maxFitnessIndex, 1, i]]]^2, {i, minTotalGate, maxTotalGate,
       1}] * Sum[If[avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]] > 0,
        0, avgTraces[[LastGoodGen, maxFitnessIndex, 2, i]]]^2,
      {i, minTotalGate, maxTotalGate, 1}])^(1/2)
```

```
0.767071

0.320424
```

Chapter 3: Ionization Model

Normalization for Continuum Radial Wave Functions

The centrifugal barrier is approximately located where the effective potential is equal to zero. Use something a bit smaller than that for the inner limit to the integration.

In[∘]:=
```
Clear[r, l, α]
```

$$\text{Solve}\left[\frac{-\alpha}{r} + \frac{l\,(l+1)}{r^2} == 0,\ r\right]$$

Out[∘]=

$$\left\{\left\{r \to \frac{l\,(1+l)}{\alpha}\right\}\right\}$$

In[∘]:= $\{\{r \to \frac{l\,(1+l)}{\alpha}\}\}$

Out[∘]= $\{\{r \to \frac{l\,(1+l)}{\alpha}\}\}$

Looks like I want to go some amount past the centrifugal barrier. Relatively more past it for low l, less for high l.

Normalization from Cowan

Normalizing continuum wave functions numerically

We are following Cowan, sec. 18-3, pg 520. In atomic units using Hartrees, we have the normalized continuum wave function as

$$P_{\epsilon l}(r) = \frac{2^{1/4}}{\pi^{1/2}\,\epsilon^{1/4}}\left[1 - \frac{1}{\epsilon\, r_0}\left(1 - \frac{5}{2\,\epsilon\, r_0} - \frac{l(l+1)}{2\, r_0}\right)\right]\frac{P_{\epsilon l}{}^{u}(r)}{B}$$

(1) Numerically integrate to large r = r_0. How large should r_0 be? According to Cowan, $r_0 > \text{Max}\left[\frac{10\,Z_c}{\epsilon}, \frac{5\,l(l+1)}{Z_c}, r_c\right]$, where r_c is the maximum radius to which the core electrons extend. Since we're dealing with Rydberg atoms, we don't really need to worry about the extent of the core electron probability density. How about the other two? For ϵ = .001 we have the following:

```
In[∘]:= Zc = 1; (* the charge of the core *)
ϵ = .001; (* continuum energy level *)
n = 20; (* principal quantum number in region of interest*)
l = 12; (* a possible value of l *)
10 Zc / ϵ
5 l (l + 1) / Zc
5 * n * (n + 15) (* max radius for discrete states *)

Out[∘]= 10 000.

Out[∘]= 780

Out[∘]= 3500
```

Based on the above, it looks like $\frac{10\,Z_c}{\epsilon}$ will be the most important...

(2) Use the numerical integrate to find the amplitude at r_0. This will be the value of B.

```
alldata = {};
ee = 0.002;
l = 1;
Zc = 1;
(*For[l=0,l≤n-1,l=l+5,*)
```

```
h = 0.0001; (* step size *)
α = 1; (* Cowan has the Coulomb potential as -2/r. I think this is a units
  issue. α is the value of the numerator in the Coulomb potential below *)
rs = Max[10 Zc/ee, 5 l (l + 1)/Zc];(*the starting point (outer limit)*)
(* ending point -- inner limit *)
(* Maybe this should be based on where the centrifugal barrier is? *)

lMin = 3;
re = If[l ≤ lMin, 1 * 9.023^(1.0/3.0),
   If[l / n ≤ 0.5, (l/(2 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0), (l/(12 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0)]];
Print[re];
x = {0, 0, 0};
g = {0, 0, 0};
r = {0, 0, 0};
t = {0, 0, 0};

data = {};

i = 1;(*this is just to make the code readable and similar to the equations*)

m = 0;
x0 = 1 * 10^-10;
x1 = 1 * 10^-5;
x[[i - 1]] = x0;
x[[i]] = x1;


r[[i - 1]] = rs;
r[[i]] = rs * e^(-1.0*h);
t[[i - 1]] = Log[r[[i - 1]]];
t[[i]] = Log[r[[i]]];
converge = {0, 0};
converged = False;
j = 2;
qd = 0;


(* calculate all parameters for loop initialization *)
r[[i + 1]] = rs * e^(-2*h);
t[[i + 1]] = Log[r[[i + 1]]];
g[[i - 1]] = 2 * e^(2.*t[[i-1]]) * (-α / r[[i - 1]] - ee) + (l + .5) * (l + .5);
```

```
g[[i]] = 2 * e^(2.*t[[i]]) * (-α / r[[i]] - ee) + (l + .5) * (l + .5);
g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - ee) + (l + .5) * (l + .5);



(*If[l>lMin,
rStop=0;,
rStop=re;
 ];*)
rStop = rs * .9;
(* The Numerov Algorithm *)
```

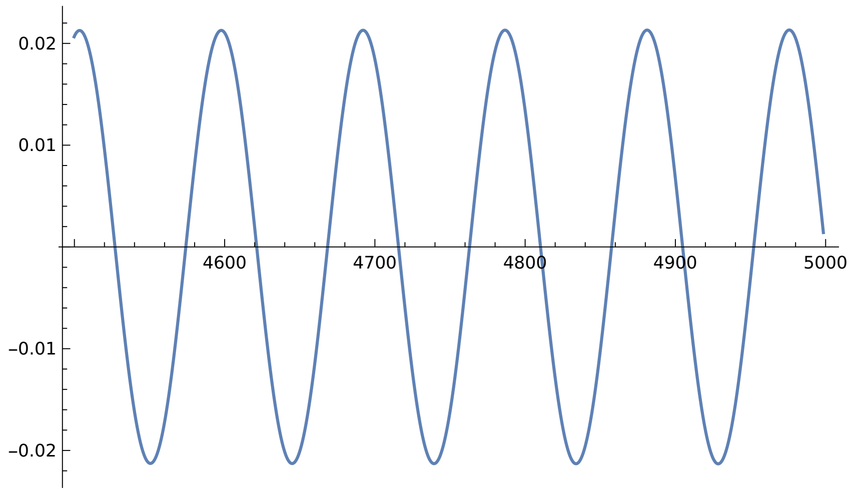
Implementing Numerov Algorithm

```
While [r[[i + 1]] > rStop,
  (* the numerov algorithm calculations *)
  x[[i + 1]] = (x[[i - 1]] * (g[[i - 1]] - 12. / h^2) + x[[i]] * (10 * g[[i]] + 24. / h^2)) /
    (12. / h^2 - g[[i + 1]]);

  (* set current points to old points and calculate new points *)
  x[[i - 1]] = x[[i]];
  x[[i]] = x[[i + 1]];

  g[[i - 1]] = g[[i]];
  g[[i]] = g[[i + 1]];

  j++; (* increment loop index *)
  r[[i + 1]] = rs * e^(-1*j*h);
  t[[i + 1]] = Log[r[[i + 1]]];
  g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - ee) + (l + .5) * (l + .5);
  AppendTo[data, {r[[i + 1]], x[[i + 1]] * r[[i + 1]]^0.5}];
  (* how to convert to the actual radial wf? *)
 ];
(*AppendTo[alldata,data];
]*)
```

Graphing Continuum Radial Wave Function

In[•]:=
```
ListLinePlot[data]
```

Out[•]=



Calculate Discrete-Discrete Matrix Elements

Calculating discrete-discrete matrix elements. Since they are discrete-discrete, they are already normalized. No normalization factor needed. Data points of {energy, matrix element} added to a table.

Loop Over Energy Given

In[•]:=
```
dataPoints = {};
convGraphs = {};
For[np = 1000, np ≤ 2000, np += 1000,
 alldata = {};
 elementTracker = {};
 h = 0.0001; (* step size *)
 threshold = 10^-8;
 qd = 0;
 σ = 1;
 n = 35;
 l = 4;
 lp = 5;
 ee = -1 / (2. * (n - qd) * (n - qd));
 eep = -1 / (2. * (np - qd) * (np - qd));
 Print["ϵn' = " <> ToString[eep]];
```

```
(*For[l=0,l≤n-1,l=l+5,*)
ns = 1/Sqrt[Abs[eep]];


α = 1; (* Cowan has the Coulomb potential as -2/r. I think this is a units
  issue. α is the value of the numerator in the Coulomb potential below *)
rs = 5 * n * (n + 15.); (*the starting point (outer limit)*)
(* ending point -- inner limit *)
(* Maybe this should be based on where the centrifugal barrier is? *)

lMin = 3;
re = If[l ≤ lMin, 1 * 9.023^(1.0/3.0),
  If[l/n ≤ 0.5, (l/(2 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0), (l/(12 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0)]];
Print[re];
(* unprimed w.f. *)
x = {0, 0, 0};
g = {0, 0, 0};
(* primed w.f. *)
xp = {0, 0, 0};
gp = {0, 0, 0};
(* coordinates *)
r = {0, 0, 0};
t = {0, 0, 0};

converge = {0, 0, 0};
element = 0;

data = {};
data2 = {};

i = 2;
(*this is just to make the code readable and similar to the equations*)

m = 0;
x0 = 10 * 10^-10;
x1 = 10 * 10^-5;
x[[i - 1]] = x0;
x[[i]] = x1;
xp[[i - 1]] = x0;
xp[[i]] = x1;
```

```
 r[[i - 1]] = rs;
r[[i]] = rs * e^(-1.0*h);
t[[i - 1]] = Log[r[[i - 1]]];
t[[i]] = Log[r[[i]]];
converge = {0, 0};
converged = False;
j = 2;
 qd = 0;


 (* calculate all parameters for loop initialization *)
 r[[i + 1]] = rs * e^(-2*h);
t[[i + 1]] = Log[r[[i + 1]]];
g[[i - 1]] = 2 * e^(2.*t[[i-1]]) * (-α / r[[i - 1]] - ee) + (l + .5) * (l + .5);
 g[[i]] = 2 * e^(2.*t[[i]]) * (-α / r[[i]] - ee) + (l + .5) * (l + .5);
g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - ee) + (l + .5) * (l + .5);

 gp[[i - 1]] = 2 * e^(2.*t[[i-1]]) * (-α / r[[i - 1]] - eep) + (lp + .5) * (lp + .5);
gp[[i]] = 2 * e^(2.*t[[i]]) * (-α / r[[i]] - eep) + (lp + .5) * (lp + .5);
gp[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - eep) + (lp + .5) * (lp + .5);

sum1 = x[[i - 1]] * xp[[i - 1]] * r[[i]]^(2+σ) + x[[i]] * xp[[i]] * r[[i]]^(2+σ);
sum2 = x[[i - 1]] * r[[i - 1]] * r[[i - 1]] + x[[i]] * r[[i]] * r[[i]];
sum3 = xp[[i - 1]] * r[[i - 1]] * r[[i - 1]] + xp[[i]] * r[[i]] * r[[i]];
element = sum1 / (sum2 * sum3)^0.5;
 converge[[i - 1]] = element;




(*If[l>lMin,
rStop=0;,
rStop=re;
  ];*)
rStop = re;
(* The Numerov Algorithm *)

 While [r[[i + 1]] > rStop && converged == False,
  (* the numerov algorithm calculations *)
   x[[i + 1]] = (x[[i - 1]] * (g[[i - 1]] - 12. / h^2) + x[[i]] * (10 * g[[i]] + 24. / h^2)) /
     (12. / h^2 - g[[i + 1]]);
   xp[[i + 1]] = (xp[[i - 1]] * (gp[[i - 1]] - 12. / h^2) + xp[[i]] * (10 * gp[[i]] + 24. / h^2)) /
```

```
      (12. / h^2 - gp[[i + 1]]);

    (* calculate contributions to matrix element *)
    sum1 += x[[i + 1]] * xp[[i + 1]] * r[[i + 1]]^(2+σ);
    sum2 += x[[i + 1]] * x[[i + 1]] * r[[i + 1]] * r[[i + 1]];
    sum3 += xp[[i + 1]] * xp[[i + 1]] * r[[i + 1]] * r[[i + 1]];

    (* set current points to old points and calculate new points *)
    x[[i - 1]] = x[[i]];
    x[[i]] = x[[i + 1]];
    xp[[i - 1]] = xp[[i]];
    xp[[i]] = xp[[i + 1]];

    g[[i - 1]] = g[[i]];
    g[[i]] = g[[i + 1]];
    gp[[i - 1]] = gp[[i]];
    gp[[i]] = gp[[i + 1]];

    j++; (* increment loop index *)
    r[[i + 1]] = rs * e^(-1*j*h);
    t[[i + 1]] = Log[r[[i + 1]]];
    g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - ee) + (l + .5) * (l + .5);
    gp[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - eep) + (lp + .5) * (lp + .5);
    element = sum1 / (sum2 * sum3)^0.5;
    converge[[i - 1]] = converge[[i]];
    converge[[i]] = element;
    If[Abs[1 - converge[[i]] / converge[[i - 1]]] ≤ threshold,
     converged = True;
    ];
    AppendTo[elementTracker, element];
    AppendTo[data, {r[[i + 1]], x[[i + 1]] * r[[i + 1]]^0.5}];
    (* how to convert to the actual radial wf? *)
    AppendTo[data2, {r[[i + 1]], xp[[i + 1]] * r[[i + 1]]^0.5}];
  ];
  element *= Sqrt[ns^3 / 2];
  Print["element = " <> ToString[element]];
  AppendTo[dataPoints, {eep, element}];
  AppendTo[convGraphs, ListLinePlot[elementTracker]];
 ]

(*AppendTo[alldata,data];
]*)
```

```
"\!\(\*SubscriptBox[\(ϵ\), \(n'\)]\) = -0.000408163"
9.953166987819678`
... Power: "Infinite expression \!\(\*FractionBox[\"1\", \"0\"]\) encountered."
"element = 1818.68"
{197.976462`, Null}
```

Out[•]= {197.976, Null}

In[•]:= dataPoints

Out[•]= {{-0.0003125, 17286.}, {-0.0002, 4690.85}, {-0.000138889, 3390.57}, {-0.000102041, 1444.26}, {-0.000078125, -3471.8}, {-0.0000617284, 2199.59}, {-0.00005, -2750.35}, {-0.0000413223, -2848.1}, {-0.0000347222, 1614.19}, {-0.0000295858, 4642.37}, {-0.0000255102, -6345.02}}

In[•]:=
```
np = 100;
eep = -1 / (2. * (np - qd) * (np - qd));
Print["ϵn' = " <> ToString[eep]];
(*For[l=0,l≤n-1,l=l+5,*)
ns = 1 / Sqrt[Abs[eep]];
Sqrt[ns^3 / 2]
```

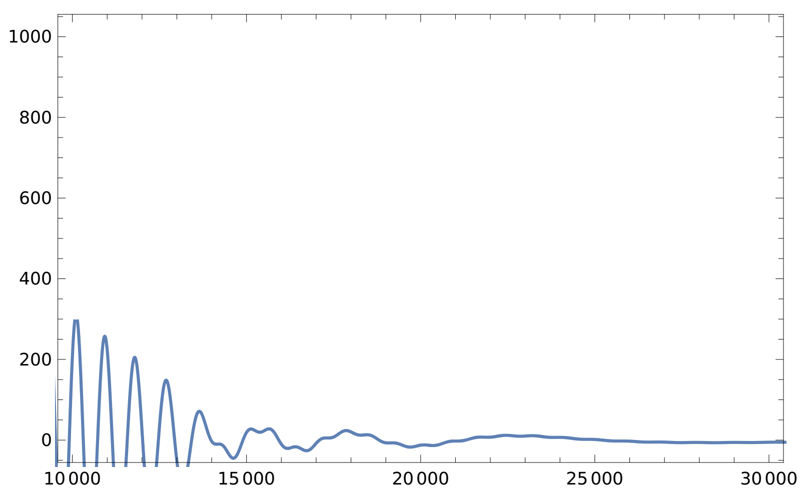
$\epsilon_{n'}$ = -0.00005

Out[•]= 1189.21

Graph to Show Convergence

In[•]:= `Show[convGraphs[[2]], PlotRange → {{10 000, 30 000}, {0, 1000}}, Frame -> True]`

Out[•]=



In[•]:= dataPoints

Out[•]= {{-0.0000125, 7467.95}, {-5.55556×10^{-6}, -18163.2}}

In[•]:= dataPoints

Out[•]= {{$-5. \times 10^{-7}$, 140219.}, {-1.25×10^{-7}, 411992.}}

Calculate Discrete-Continuum Matrix Elements

First use the file NumerovContinuumRadialWF_normalization.nb to calculate the normalization constant for the value of ϵ below.

```
Timing[
 alldata = {};
 elementTracker = {};
 h = 0.0001; (* step size *)
 threshold = 10^-10;
 σ = 1;
 n = 35;
 l = 4;
 lp = 5;
 ee = -1 / (2. * (n - qd) * (n - qd));
 eep = 0.002;
 (* choose the appropriate normalization for your value of eep,
 the energy of the continuum wf *)
 norm = 2.0818542390318244`;
 (*For[l=0,l≤n-1,l=l+5,*)


 α = 1; (* Cowan has the Coulomb potential as -2/r. I think this is a units
   issue. α is the value of the numerator in the Coulomb potential below *)
 rs = 5 * n * (n + 15.); (*the starting point (outer limit)*)
 (* ending point -- inner limit *)
 (* Maybe this should be based on where the centrifugal barrier is? *)

lMin = 3;
 re = If[l ≤ lMin, 1 * 9.023^(1.0/3.0),
   If[l / n ≤ 0.5, (l/(2 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0), (l/(12 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0)]];
 Print[re];
 (* unprimed w.f. *)
 x = {0, 0, 0};
g = {0, 0, 0};
 (* primed w.f. *)
 xp = {0, 0, 0};
gp = {0, 0, 0};
```

```
 (* coordinates *)
 r = {0, 0, 0};
 t = {0, 0, 0};

 converge = {0, 0, 0};
 element = 0;

 data = {};
data2 = {};

 i = 2;
 (*this is just to make the code readable and similar to the equations*)

m = 0;
x0 = 10 * 10^-10;
 x1 = 10 * 10^-5;
 x[[i - 1]] = x0;
x[[i]] = x1;
xp[[i - 1]] = x0;
xp[[i]] = x1;



r[[i - 1]] = rs;
r[[i]] = rs * e^(-1.0*h);
t[[i - 1]] = Log[r[[i - 1]]];
t[[i]] = Log[r[[i]]];
converge = {0, 0};
converged = False;
j = 2;
 qd = 0;


 (* calculate all parameters for loop initialization *)
r[[i + 1]] = rs * e^(-2*h);
t[[i + 1]] = Log[r[[i + 1]]];
g[[i - 1]] = 2 * e^(2.*t[[i-1]]) * (-α / r[[i - 1]] - ee) + (l + .5) * (l + .5);
 g[[i]] = 2 * e^(2.*t[[i]]) * (-α / r[[i]] - ee) + (l + .5) * (l + .5);
g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - ee) + (l + .5) * (l + .5);

 gp[[i - 1]] = 2 * e^(2.*t[[i-1]]) * (-α / r[[i - 1]] - eep) + (lp + .5) * (lp + .5);
gp[[i]] = 2 * e^(2.*t[[i]]) * (-α / r[[i]] - eep) + (lp + .5) * (lp + .5);
gp[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - eep) + (lp + .5) * (lp + .5);
```

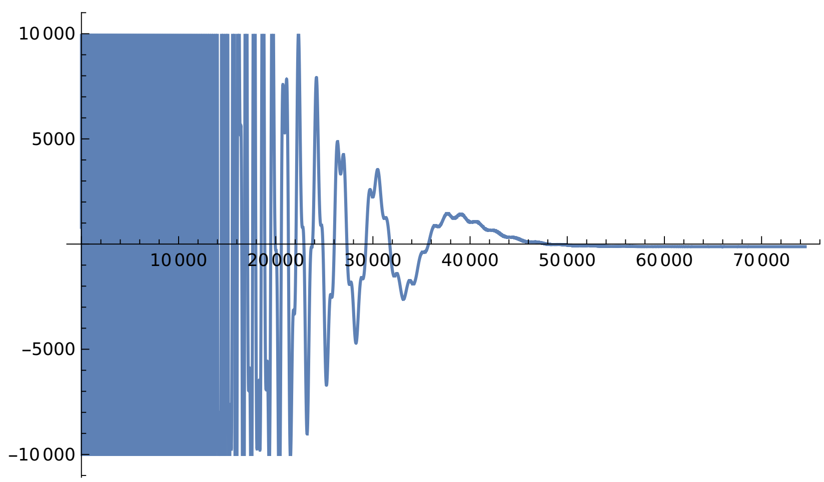
```
sum1 = x[[i - 1]] * xp[[i - 1]] * r[[i]]^3 + x[[i]] * xp[[i]] * r[[i]]^(2+σ);
sum2 = x[[i - 1]] * r[[i - 1]] * r[[i - 1]] + x[[i]] * r[[i]] * r[[i]];
sum3 = xp[[i - 1]] * r[[i - 1]] * r[[i - 1]] + xp[[i]] * r[[i]] * r[[i]];
element = norm * sum1 / (sum2)^0.5;
 converge[[i - 1]]  = element;




 (*If[l>lMin,
rStop=0;,
rStop=re;
  ];*)
rStop = re;
 (* The Numerov Algorithm *)

 While [r[[i + 1]] > rStop && converged == False,
  (* the numerov algorithm calculations *)
  x[[i + 1]] = (x[[i - 1]] * (g[[i - 1]] - 12. / h^2) + x[[i]] * (10 * g[[i]] + 24. / h^2)) /
    (12. / h^2 - g[[i + 1]]);
  xp[[i + 1]] = (xp[[i - 1]] * (gp[[i - 1]] - 12. / h^2) + xp[[i]] * (10 * gp[[i]] + 24. / h^2)) /
    (12. / h^2 - gp[[i + 1]]);

  (* calculate contributions to matrix element *)
  sum1 += x[[i + 1]] * xp[[i + 1]] * r[[i + 1]]^(2+σ);
  sum2 += x[[i + 1]] * x[[i + 1]] * r[[i + 1]] * r[[i + 1]];
  sum3 += xp[[i + 1]] * xp[[i + 1]] * r[[i + 1]] * r[[i + 1]];

  (* set current points to old points and calculate new points *)
  x[[i - 1]] = x[[i]];
  x[[i]] = x[[i + 1]];
  xp[[i - 1]] = xp[[i]];
  xp[[i]] = xp[[i + 1]];

  g[[i - 1]] = g[[i]];
  g[[i]] = g[[i + 1]];
  gp[[i - 1]] = gp[[i]];
  gp[[i]] = gp[[i + 1]];

  j++; (* increment loop index *)
  r[[i + 1]] = rs * e^(-1*j*h);
  t[[i + 1]] = Log[r[[i + 1]]];
```

```
  g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - ee) + (l + .5) * (l + .5);
  gp[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - eep) + (lp + .5) * (lp + .5);
  element = norm * sum1 / (sum2)^0.5;
  converge[[i - 1]] = converge[[i]];
  converge[[i]] = element;
  If[Abs[1 - converge[[i]] / converge[[i - 1]]] ≤ threshold,
   converged = True;
  ];
  AppendTo[elementTracker, element];
  AppendTo[data, {r[[i + 1]], x[[i + 1]] * r[[i + 1]]^0.5}];
  (* how to convert to the actual radial wf? *)
  AppendTo[data2, {r[[i + 1]], xp[[i + 1]] * r[[i + 1]]^0.5}];
 ];]
(*AppendTo[alldata,data];
]*)
```

Graph Element

In[∘]:= `ListLinePlot[elementTracker[[1 ;;]]]`

Out[∘]=



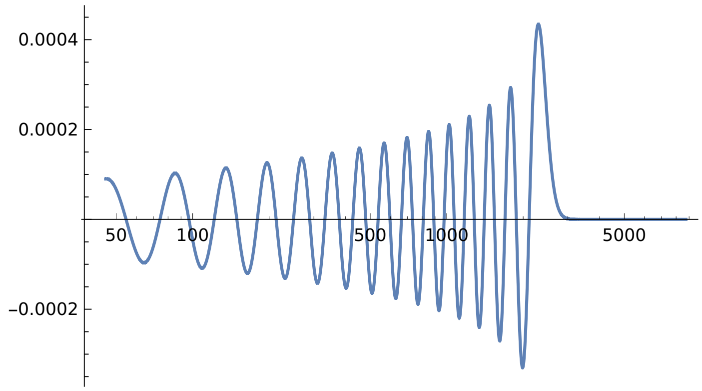
The value of the matrix element.

In[∘]:= `element`

Out[∘]= 3.23297×10^{6}

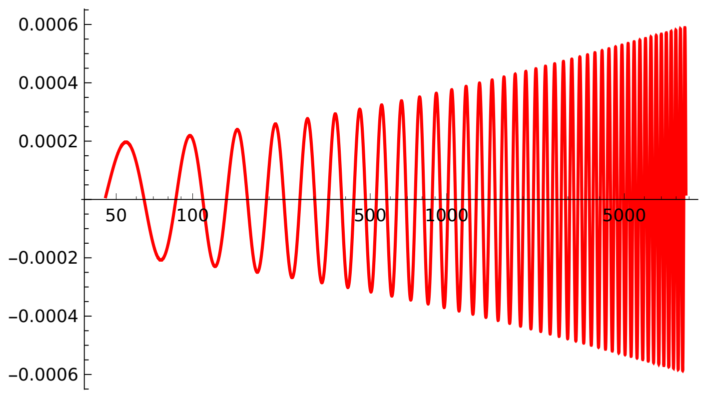
In[◦]:=
```
a = ListLogLinearPlot[{#[[1]], #[[2]] / sum2^.5} & /@ data,
  Joined → True, PlotRange → {All, All}]
```

Out[◦]=



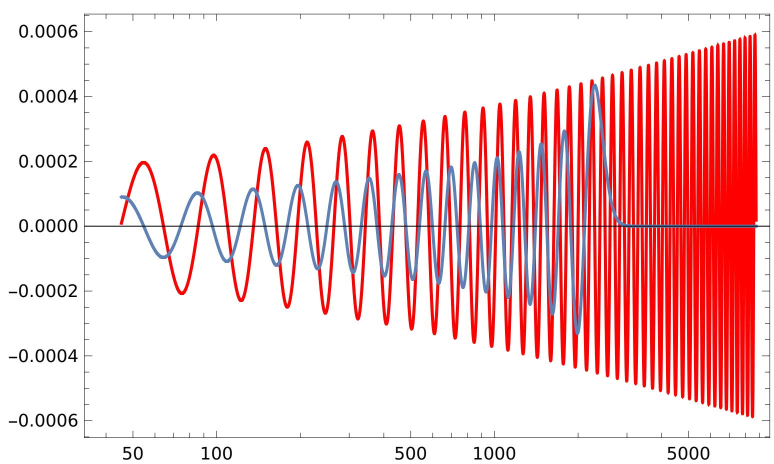
In[◦]:=
```
b = ListLogLinearPlot[{#[[1]], 0.001 * #[[2]]} & /@ data2,
  Joined → True, PlotRange → {All, All}, PlotStyle → Red]
```

Out[◦]=



In[◦]:=
```
Show[b, a, Frame → True, PlotRange → All]
```

Out[◦]=



In[◦]:= $2^{.25}\,\pi^{-.5}\,\mathrm{eep}^{-.25}$

Out[◦]= 3.77296

In[•]:= **converged**

Out[•]= False

In[•]:= **Length[alldata]**

Out[•]= 7

In[•]:= **alldata[[2]]**

Out[•]= {}

In[•]:= **Last[data]**

... Last: {} has zero length and no last element.

Out[•]= Last[{}]

Length[data]

42 296

j

42 298

In[•]:= **rStop**

Out[•]= 688.342

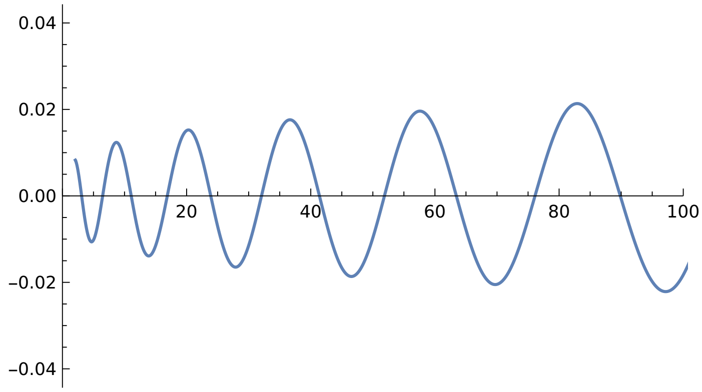
rs * ⅇ^(-1*j*h)

1.80452×10^{-181}

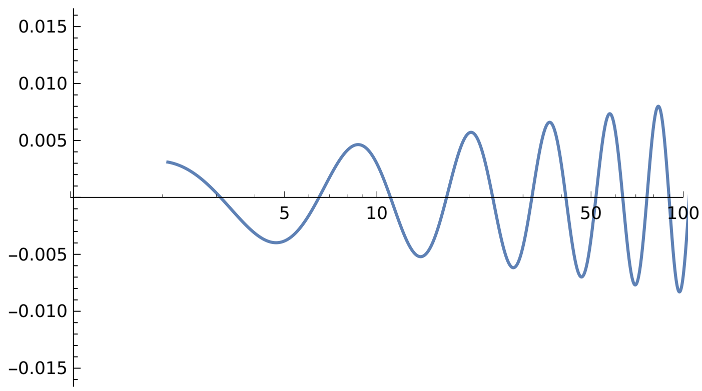
data[[360 ;; 380]]

```
{{24.1044, 23.4162}, {23.8646, 19.839}, {23.6271, 16.3542}, {23.392, 12.941},
 {23.1593, 9.57828}, {22.9288, 6.24499}, {22.7007, 2.91966}, {22.4748, -0.419592},
 {22.2512, -3.79525}, {22.0298, -7.23054}, {21.8106, -10.7496},
 {21.5936, -14.3777}, {21.3787, -18.1412}, {21.166, -22.0683},
 {20.9554, -26.1887}, {20.7469, -30.5339}, {20.5404, -35.1378},
 {20.336, -40.0369}, {20.1337, -45.2703}, {19.9334, -50.8802}, {19.735, -56.9127}}
```

In[•]:= `ListLinePlot[data, PlotRange → {{0, 100}, All}]`

Out[•]=


In[•]:= `ListLogLinearPlot[data, Joined → True, PlotRange → {{1, 100}, All}]`

Out[•]=


In[•]:= $1 / .01^{.25}$

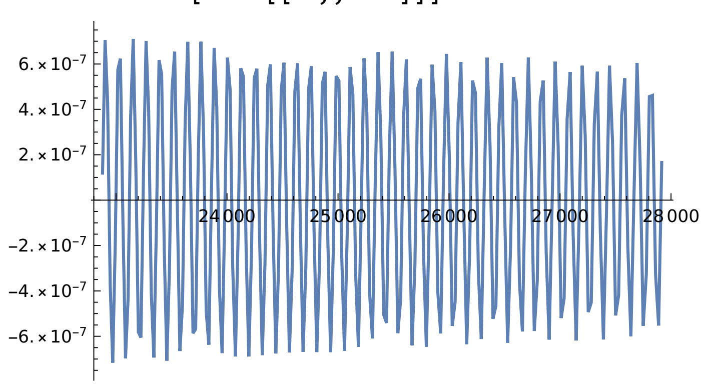
Out[•]= 3.16228

In[•]:= `Table[ListLogLinearPlot[alldata[[i]], Joined → True, PlotRange → {{1, 4000}, All}], {i, 1, Length[alldata]}]`

In[•]:= `re`

Out[•]= 1957.47

`ListLinePlot[data[[1 ;; 200]]]`



```
Length[data]
```

7089

```
re
```

23.3168

```
Last[data]
```

{270.428, -133.376}

```
data[[1 ;; 100]]
```

In[∘]:=
```
-120.0 / (2 * 109 737.3)
```

Out[∘]= -0.00054676

In[∘]:=
$$\left(\frac{2^{0.5}}{0.000546760308482166`^{1.5}}\right)^{1} * 700$$

Out[∘]= 7.74315×10^{7}

Automated Matrix Element

Combining discrete-continuum matrix elements document and normalization document to have one automated file. Automatically finds normalization and applies it to each corresponding energy.

Finding the Normalization

In[∘]:=
```
findNormalization[eep_, l_] := Module[{Zc, h, α, rs, lMin, re, n, x, g,
    r, t, data, i, m, x0, x1, converge, converged, j, qd, B, rstop, norm},
  norm[r_, ee_] := 2^0.25/(π^0.5 ee^0.25 B) (1 - 1/(ee r) (1 - 5/(2 ee r) - l (l + 1)/(2 r)));
  Zc = 1;
  (*For[l=0,l≤n-1,l=l+5,*)

  h = 0.0001; (* step size *)
  α = 1; (* Cowan has the Coulomb potential as -2/r. I think this is a units
     issue. α is the value of the numerator in the Coulomb potential below *)
  rs = Max[10 Zc/eep, 5 l (l + 1)/Zc]; (*the starting point (outer limit)*)
  (* ending point -- inner limit *)
(* Maybe this should be based on where the centrifugal barrier is? *)
```

```
    lMin = 3;
    n = 35; (* do something smarter here *)
    re = If[l ≤ lMin, 1 * 9.023^(1.0/3.0),
      If[l / n ≤ 0.5, (l/(2 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0), (l/(12 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0)]];
    x = {0, 0, 0};
g = {0, 0, 0};
    r = {0, 0, 0};
    t = {0, 0, 0};

    data = {};

    i = 1;
    (*this is just to make the code readable and similar to the equations*)

m = 0;
x0 = 1 * 10^-10;
    x1 = 1 * 10^-5;
    x[[i - 1]] = x0;
x[[i]] = x1;


r[[i - 1]] = rs;
r[[i]] = rs * e^(-1.0*h);
t[[i - 1]] = Log[r[[i - 1]]];
t[[i]] = Log[r[[i]]];
converge = {0, 0};
converged = False;
j = 2;
    qd = 0;


(* calculate all parameters for loop initialization *)
    r[[i + 1]] = rs * e^(-2*h);
t[[i + 1]] = Log[r[[i + 1]]];
g[[i - 1]] = 2 * e^(2.*t[[i-1]]) * (-α / r[[i - 1]] - eep) + (l + .5) * (l + .5);
g[[i]] = 2 * e^(2.*t[[i]]) * (-α / r[[i]] - eep) + (l + .5) * (l + .5);
g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - eep) + (l + .5) * (l + .5);



(*If[l>lMin,
```

```
rStop=0;,
rStop=re;
     ];*)
    rStop = rs * .9;
    (* The Numerov Algorithm *)

While [r[[i + 1]] > rStop,
     (* the numerov algorithm calculations *)
     x[[i + 1]] = (x[[i - 1]] * (g[[i - 1]] - 12. / h^2) + x[[i]] * (10 * g[[i]] + 24. / h^2)) /
       (12. / h^2 - g[[i + 1]]);

     (* set current points to old points and calculate new points *)
     x[[i - 1]] = x[[i]];
     x[[i]] = x[[i + 1]];

     g[[i - 1]] = g[[i]];
     g[[i]] = g[[i + 1]];

     j++; (* increment loop index *)
     r[[i + 1]] = rs * e^(-1*j*h);
     t[[i + 1]] = Log[r[[i + 1]]];
     g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - eep) + (l + .5) * (l + .5);
     AppendTo[data, {r[[i + 1]], x[[i + 1]] * r[[i + 1]]^0.5}];
    ];
B = Max[#[[2]] & /@ data];
    Return[norm[rs, eep]];
   ];
```

Applying Normalization to Discrete-Continuum Matrix Elements

```
dataPoints = {};
convGraphs = {};
For[eep = 0.00001, eep ≤ 0.0001, eep += 0.000005,

(*matrix element*)

 elementTracker = {};
 h = 0.0001; (* step size *)
 threshold = 10^-10;
 σ = 1;
 n = 35;
 l = 4;
 lp = 5;
```

```
ee = -1 / (2. * (n - qd) * (n - qd));
NORM = findNormalization[eep, lp];
Print["normalization = " <> ToString[NORM]];
ns = 1 / Sqrt[Abs[eep]];


α = 1; (* Cowan has the Coulomb potential as -2/r. I think this is a units
  issue. α is the value of the numerator in the Coulomb potential below *)
rs = 5 * n * (n + 15.); (*the starting point (outer limit)*)
(* ending point -- inner limit *)
(* Maybe this should be based on where the centrifugal barrier is? *)

lMin = 3;
re = If[l ≤ lMin, 1 * 9.023^(1.0/3.0),
  If[l / n ≤ 0.5, (l/(2 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0), (l/(12 n))^.5 (l (1 + l))/α * 9.023^(1.0/3.0)]];
(* unprimed w.f. *)
x = {0, 0, 0};
g = {0, 0, 0};
(* primed w.f. *)
xp = {0, 0, 0};
gp = {0, 0, 0};
(* coordinates *)
r = {0, 0, 0};
t = {0, 0, 0};

converge = {0, 0, 0};
element = 0;

data = {};
data2 = {};

i = 2;
(*this is just to make the code readable and similar to the equations*)

m = 0;
x0 = 10 * 10^-10;
x1 = 10 * 10^-5;
x[[i - 1]] = x0;
x[[i]] = x1;
xp[[i - 1]] = x0;
xp[[i]] = x1;
```

```
r[[i - 1]] = rs;
r[[i]] = rs * e^(-1.0*h);
t[[i - 1]] = Log[r[[i - 1]]];
t[[i]] = Log[r[[i]]];
converge = {0, 0};
converged = False;
j = 2;
 qd = 0;


 (* calculate all parameters for loop initialization *)
 r[[i + 1]] = rs * e^(-2*h);
t[[i + 1]] = Log[r[[i + 1]]];
g[[i - 1]] = 2 * e^(2.*t[[i-1]]) * (-α / r[[i - 1]] - ee) + (l + .5) * (l + .5);
 g[[i]] = 2 * e^(2.*t[[i]]) * (-α / r[[i]] - ee) + (l + .5) * (l + .5);
g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - ee) + (l + .5) * (l + .5);

 gp[[i - 1]] = 2 * e^(2.*t[[i-1]]) * (-α / r[[i - 1]] - eep) + (lp + .5) * (lp + .5);
gp[[i]] = 2 * e^(2.*t[[i]]) * (-α / r[[i]] - eep) + (lp + .5) * (lp + .5);
gp[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - eep) + (lp + .5) * (lp + .5);

 sum1 = x[[i - 1]] * xp[[i - 1]] * r[[i]]^3 + x[[i]] * xp[[i]] * r[[i]]^(2+σ);
sum2 = x[[i - 1]] * r[[i - 1]] * r[[i - 1]] + x[[i]] * r[[i]] * r[[i]];
sum3 = xp[[i - 1]] * r[[i - 1]] * r[[i - 1]] + xp[[i]] * r[[i]] * r[[i]];
element = NORM * sum1 / (sum2)^0.5;
 converge[[i - 1]] = element;




 (*If[l>lMin,
rStop=0;,
rStop=re;
  ];*)
 rStop = re;
 (* The Numerov Algorithm *)

While [r[[i + 1]] > rStop && converged == False,
  (* the numerov algorithm calculations *)
  x[[i + 1]] = (x[[i - 1]] * (g[[i - 1]] - 12. / h^2) + x[[i]] * (10 * g[[i]] + 24. / h^2)) /
    (12. / h^2 - g[[i + 1]]);
```

```
   xp[[i + 1]] = (xp[[i - 1]] * (gp[[i - 1]] - 12. / h^2) + xp[[i]] * (10 * gp[[i]] + 24. / h^2)) /
     (12. / h^2 - gp[[i + 1]]);

   (* calculate contributions to matrix element *)
   sum1 += x[[i + 1]] * xp[[i + 1]] * r[[i + 1]]^(2+σ);
   sum2 += x[[i + 1]] * x[[i + 1]] * r[[i + 1]] * r[[i + 1]];
   sum3 += xp[[i + 1]] * xp[[i + 1]] * r[[i + 1]] * r[[i + 1]];

   (* set current points to old points and calculate new points *)
   x[[i - 1]] = x[[i]];
   x[[i]] = x[[i + 1]];
   xp[[i - 1]] = xp[[i]];
   xp[[i]] = xp[[i + 1]];

   g[[i - 1]] = g[[i]];
   g[[i]] = g[[i + 1]];
   gp[[i - 1]] = gp[[i]];
   gp[[i]] = gp[[i + 1]];

   j++; (* increment loop index *)
   r[[i + 1]] = rs * e^(-1*j*h);
   t[[i + 1]] = Log[r[[i + 1]]];
   g[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - ee) + (l + .5) * (l + .5);
   gp[[i + 1]] = 2 * e^(2.*t[[i+1]]) * (-α / r[[i + 1]] - eep) + (lp + .5) * (lp + .5);
   element = NORM * sum1 / (sum2)^0.5;
   converge[[i - 1]] = converge[[i]];
   converge[[i]] = element;
   If[Abs[1 - converge[[i]] / converge[[i - 1]]] ≤ threshold,
    converged = True;
   ];
   AppendTo[elementTracker, element];
   AppendTo[data, {r[[i + 1]], x[[i + 1]] * r[[i + 1]]^0.5}];
   (* how to convert to the actual radial wf? *)
   AppendTo[data2, {r[[i + 1]], xp[[i + 1]] * r[[i + 1]]^0.5}];
  ];
  element /= Sqrt[ns^3 / 2];
  AppendTo[dataPoints, {eep, element}];
  Print["For energy = " <> ToString[eep] <>
    ", the matrix element = " <> ToString[element]];
  AppendTo[convGraphs, ListLinePlot[elementTracker]];
 ]
9.953166987819678`
... Power: "Infinite expression \!\(\*FractionBox[\"1\", \"0\"]\) encountered."
```

Updated Output Showing Energy, Matrix Element and Normalization

```
For energy = 0.00002, the matrix element = 100.227
normalization = 405.827
For energy = 0.000025, the matrix element = -2287.14
normalization = 388.619
For energy = 0.00003, the matrix element = 1076.99
normalization = 374.66
For energy = 0.000035, the matrix element = 2064.59
normalization = 362.912
For energy = 0.00004, the matrix element = -2078.17
normalization = 352.651
For energy = 0.000045, the matrix element = -1424.52
normalization = 343.816
For energy = 0.00005, the matrix element = 2718.59
normalization = 335.888
For energy = 0.000055, the matrix element = 649.772
normalization = 328.84
For energy = 0.00006, the matrix element = -3029.43
normalization = 322.536
For energy = 0.000065, the matrix element = 72.958
normalization = 316.672
For energy = 0.00007, the matrix element = 3111.53
normalization = 311.338
For energy = 0.000075, the matrix element = -633.737
normalization = 306.566
For energy = 0.00008, the matrix element = -3085.05
normalization = 301.932
For energy = 0.000085, the matrix element = 1023.07
normalization = 297.686
For energy = 0.00009, the matrix element = -1327.6
normalization = 293.837
For energy = 0.000095, the matrix element = -1239.21
normalization = 290.108
For energy = 0.0001, the matrix element = -3012.07
```

Out[∘]= 9.95317

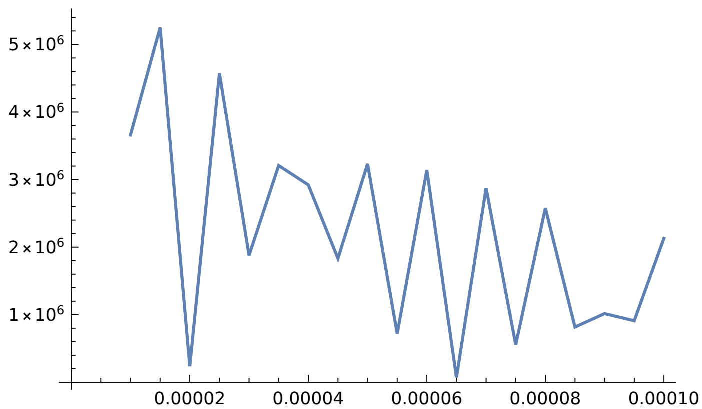
Graph of Data Points

In[∘]:=
```
dataPoints
```

In[∘]:=
```
dataPoints = {{0.00001`, -3.6611103203745643`*^6},
   {0.000015000000000000002`, 5.251014611151936`*^6},
   {0.00002`, 236973.06751533173`}, {0.000025`, -4.574270960365067`*^6},
   {0.00003`, 1.878688221083168`*^6}, {0.000035000000000000004`,
    3.2082463347520446`*^6}, {0.00004`, -2.9216016605495047`*^6},
   {0.000045`, -1.8333507432154668`*^6}, {0.00005`, 3.232965799173056`*^6},
   {0.000055`, 719406.230104656`}, {0.00006`, -3.142191531993445`*^6},
   {0.00006500000000000001`, 71264.45621315317`},
   {0.00007000000000000001`, 2.8749832067233287`*^6},
   {0.00007500000000000001`, -556030.2025240052`},
   {0.00008`, -2.5788747650883086`*^6},
   {0.000085`, 817193.8174503817`}, {0.00009`, -1.0159475401556246`*^6},
   {0.000095`, -910620.9480040356`}, {0.0001`, -2.129852926667158`*^6}};
```

In[∘]:=
```
ListPlot[Abs[dataPoints], Joined → True]
```

Out[∘]=

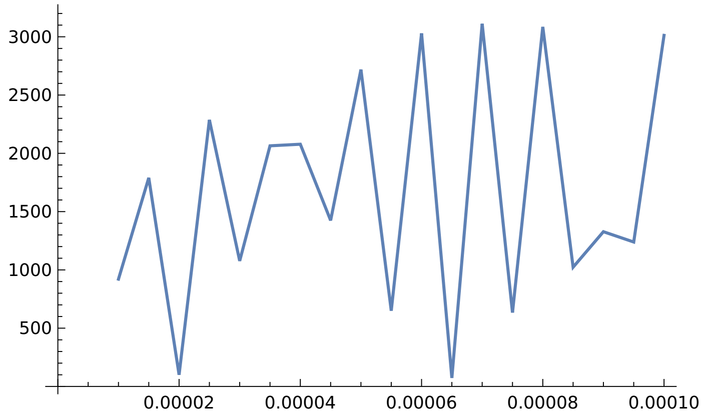


In[∘]:=
```
dataPoints
```

Out[∘]=
```
{{0.00001, -920.721}, {0.000015, 1789.89}, {0.00002, 100.227}, {0.000025, -2287.14},
 {0.00003, 1076.99}, {0.000035, 2064.59}, {0.00004, -2078.17}, {0.000045, -1424.52},
 {0.00005, 2718.59}, {0.000055, 649.772}, {0.00006, -3029.43}, {0.000065, 72.958},
 {0.00007, 3111.53}, {0.000075, -633.737}, {0.00008, -3085.05},
 {0.000085, 1023.07}, {0.00009, -1327.6}, {0.000095, -1239.21}, {0.0001, -3012.07}}
```

In[•]:= `ListPlot[Abs[dataPoints], Joined → True]`

Out[•]=



References

[1] Rachel Feynman, Jacob Hollingsworth, Michael Vennettilli, Tamas Budner, Ryan Zmiewski, Donald P. Fahey, Thomas J. Carroll, and Michael W. Noel. Quantum interference in the field ionization of Rydberg atoms. Phys. Rev. A, 92(4):043412, October 2015.

[2] Myron L. Zimmerman, Michael G. Littman, Michael M. Kash, and Daniel Kleppner. Stark structure of the Rydberg states of alkali-metal atoms. Phys. Rev. A, 20(6):2251–2275, December 1979.

[3] Robert D. Cowan. The Theory of Atomic Structure and Spectra. University of California Press, September 1981.

[4] Jan R. Rubbmark, Michael M. Kash, Michael G. Littman, and Daniel Kleppner. Dynamical effects at avoided level crossings: A study of the Landau-Zener effect using Rydberg atoms. Phys. Rev. A, 23(6):3107–3117, June 1981.

[5] Emily Altiere, Donald P. Fahey, Michael W. Noel, Rachel J. Smith, and Thomas J. Carroll. Dipole-dipole interaction between rubidium Rydberg atoms. Phys. Rev. A, 84(5):053431, November 2011.

[6] J. H. Hoogenraad and L. D. Noordam. Rydberg atoms in far-infrared radiation fields. I. Dipole matrix elements of H, Li, and Rb. Phys. Rev. A, 57(6):4533–4545, June 1998.

[7] Thomas F. Gallagher. Rydberg Atoms. Cambridge University Press, Cambridge ; New York, October 1994.

[8] Vincent C. Gregoric, Xinyue Kang, Zhimin Cheryl Liu, Zoe A. Rowley, Thomas J. Carroll, and Michael W. Noel. Quantum control via a genetic algorithm of the field ionization pathway of a Rydberg electron. Phys. Rev. A, 96(2):023403, August 2017.