Classical Zeno Effect Seen in the Coherence of Light Sources

In this notebook, we wish to repeat the results of [1]. In this paper, the authors perform an example of the Zeno effect in the classical limit, showing that use of slits to cause a distrubance on a beam of light can cause the measured intensity over a set distance to increase.

To do this, we have 4 equations we need to find:

1. $J(x_1, x_2) = \langle E(x_1)E^*(x_2)\rangle = \frac{1}{2a}exp\left[\frac{(x_1-x_2)^2}{d^2}\right]$
2. $P = \int_{-a}^{a} J(x,x)dx$
3. $\mu_g = \frac{1}{P} \cdot \left[\int\int_{-a}^{a} |J(x_1,x_2)|^2\right]^{1/2}$
4. $J(x_1', x_2') = \frac{k}{2\pi z} \cdot \int\int_{-a}^{a} J(x_1,x_2) \times K(x_1' - x_1)K^*(x_2' - x_2)dx_1dx_2$

Where:
$K(x) = exp\left[\frac{ikx^2}{2z}\right]$

We solve this set of equation numerically.

[1] M.A. Porras, A. Luis, and I. Gonzalo, Classical Zeno dynamics in the light emitted by an extended, partially coherent source, Phys. Rev. A 88, 052101

```
Main.workspace3.Data
```

```
Main.workspace3.calculatePower
```

```julia
"""
Impliments equation (2)
"""
function calculatePower(dat::Data)
    power = 0

    for i in 1:dat.n-1
        power += dat.a[i,i] * dat.step
    end
    #This currently can return a complex number. Is this correct?
    return Float64(abs(power))
end
```

```
Main.workspace3.calculateCoherence
```

```julia
"""
Impliments equation (3)
"""
function calculateCoherence(dat::Data, power)
```

```
    coh = 0;

    for i in 1:dat.n
        for j in 1:dat.n
            #
            coh += abs(dat.a[i,j])^2 * 2 * dat.step^2
        end
    end

    coh = Float64(sqrt(coh)/abs(power))
    return coh;
end
```

getCentreIntensity (generic function with 1 method)

```
function getCentreIntensity(dat::Data)
    return abs(dat.a[convert(Int64, (dat.n+1)/2), convert(Int64, (dat.n+1)/2)])
end
```

initExperimentValues (generic function with 1 method)

```
function initExperimentValues(dat::Data)
    for i in 1:dat.n
        dat.x[i] = -1 + 2 * (i-1)/(dat.n-1);
    end

    for i in 1:dat.n
        for j in 1:dat.n

            #dat.a = J(x1,x2) -> Equation 1 (a=1)
            dat.a[i,j] = exp(- (dat.x[i] - dat.x[j])^2 / dat.dd^2)/2

            #Equation for K(x) and K*(x)
            #We have also included the numberical
            dat.b[i,j] = exp(-1im * (dat.x[i]-dat.x[j])^2 / (2*dat.z)) * dat.ss
            dat.d[i,j] = conj(dat.b[i,j])
        end
    end
end
```

Main.workspace3.iterateOverSlits

```
"""
Runs the calculation, iterating over all slits within this experiment to
calculate the final result at the detector
"""
function iterateOverSlits(dat::Data, print_all=false)
    #iterate all slits
    for m in 1:dat.nr

        #Fill the first octant of e
        for i in 1:convert(Int64, (dat.n+1)/2)
            for j in 1:i
                dat.e[i,j] = 0
                for k in 1:dat.n
                    dat.c[k,j] = 0
                    for l in 1:dat.n
                        dat.c[k,j ] += dat.a[k,l] * dat.b[l,j]
                    end
                    dat.e[i,j] += dat.d[i,k] * dat.c[k,j]
                end
            end
        end

        #Fill the octant below
        for i in convert(Int64, (dat.n+3)/2):dat.n
            for j in 1:(dat.n-i+1)
                dat.e[i,j] = 0
                for k in 1:dat.n
```

```
                    dat.c[k,j] = 0
                    for l in 1:dat.n
                        dat.c[k,j] += dat.a[k,l] * dat.b[l,j]
                    end
                    dat.e[i,j] += dat.d[i,k]* dat.c[k,j]
                end
            end
        end

        #Fill the final quadrants
        for i in convert(Int64, (dat.n+3)/2):dat.n
            for j in (dat.n-i+2):i
                dat.e[i,j] = conj(dat.e[dat.n+1-j, dat.n+1-i])
            end
        end

        #finalising data
        for i in 1:dat.n
            for j in 1:dat.n
                if( j<= i)
                    dat.a[i,j] = dat.e[i,j]
                else
                    dat.a[i,j] = conj(dat.e[j,i])
                end
            end
        end
    end

    return
end

```

Main.workspace3.runExperiment

```
"""
Runs the experiment on the supplied data. We first initialise this data.
We then calculate the relevent start values, before running the calculation by
calling 'iterateOverSlits'.
Once this is complete, we re-calculate relevent end values before returning all.
"""
function runExperiment(dat::Data)
    #Set up
    initExperimentValues(dat)

    #calculate start values
    startPower = calculatePower(dat)
    startCoh = calculateCoherence(dat, startPower)
    startIntensity = getCentreIntensity(dat)
    startA = copy(dat.a)

    #run calculation
    iterateOverSlits(dat)

    #calculate end values
    endPower = calculatePower(dat)
    endCoh = calculateCoherence(dat, endPower)
    endIntensity = getCentreIntensity(dat)
    endA = copy(dat.a)

    return startPower, startCoh, startIntensity, endPower, endCoh, endIntensity,
startA, endA

end
```

Main.workspace3.runMultipleExperiments

```
"""
Runs a set of different experiments, with all varaibles constant except
the total number of slits 'nr'.
```

```
# Arguments
- 'n::Integer' : The dimensionality to solve over
- 'zmax::Float' : The distance between soruce slit and detector
- 'dd::Float' : Not really sure
"""
function runMultipleExperiments(n, zmax, dd, min_nr, max_nr)

    results = zeros(max_nr-min_nr + 1, 7) * 1im
    #iterate for different slit counts
    for nr in min_nr:max_nr
        #create data with this nr
        dat = Data(n,zmax, nr, dd)

        #run calculation
        sPow, sCoh, sInt, ePow, eCoh, eInt = runExperiment(dat)

        #store all values
        results[nr - min_nr + 1, 1] = nr
        results[nr - min_nr + 1, 2] = sPow;
        results[nr - min_nr + 1, 3] = sCoh;
        results[nr - min_nr + 1, 4] = sInt;

        results[nr - min_nr + 1, 5] = ePow;
        results[nr - min_nr + 1, 6] = eCoh;
        results[nr - min_nr + 1, 7] = eInt;

    end
    return results

end
```

```
begin
    struct GlobalArgs
        n; zmax; dd;
    end
end
```

```
exp_1_args = GlobalArgs(51, 0.5, 0.1)
```

```
min_max_nr_exp_1 = (1, 10)
```

☑

```
@bind run_exp_1 CheckBox()
```

Press the toggle button above to start the calculation

```
"Experiment 1 run succesfully for slit counts between 1 and (1, 10)[2]"
```

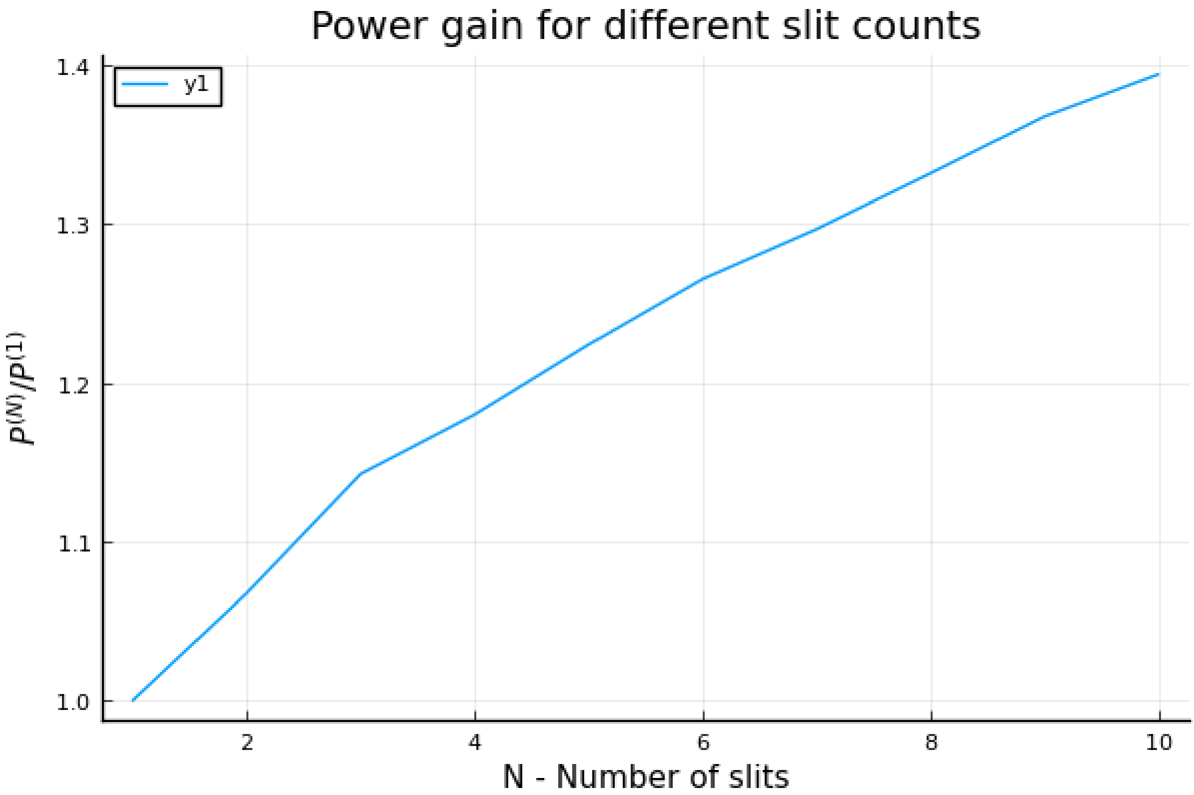
We have generated and stored our results as a 2d array. Each row represents a different experiment, with the collomns representing (from left to right):

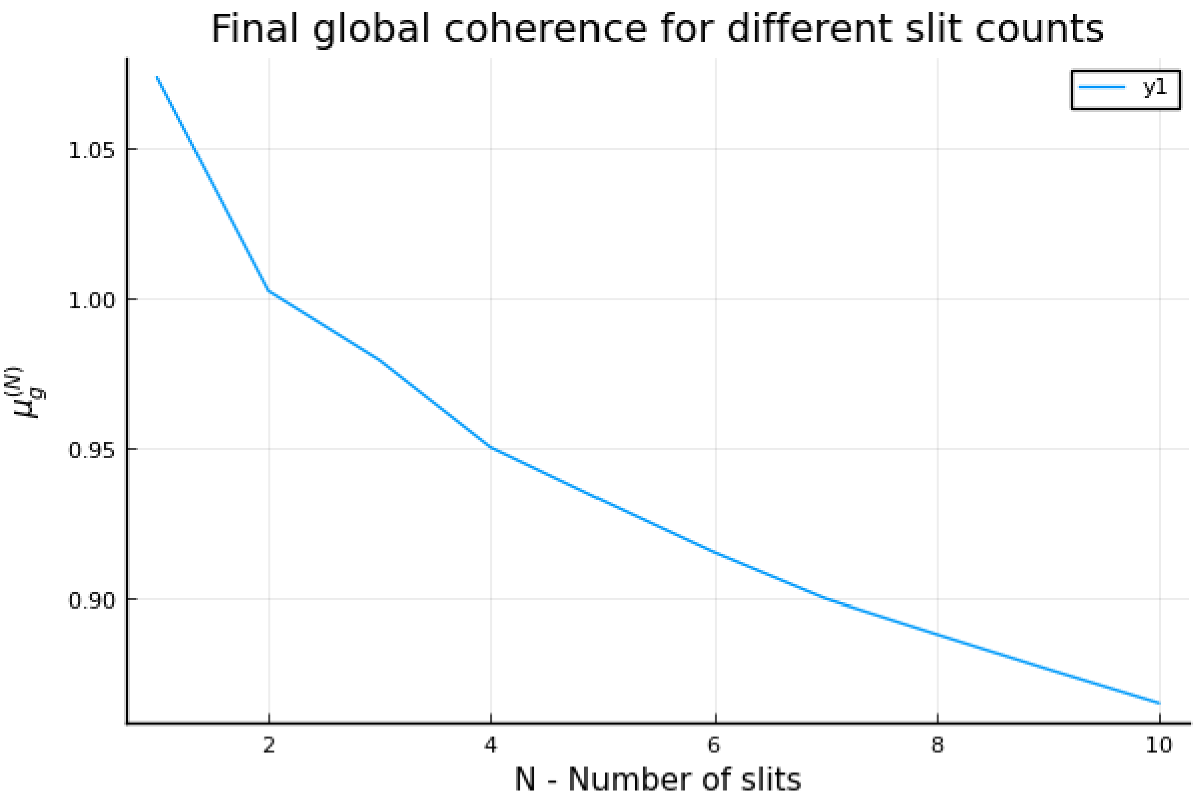
1. NR - The number of slits for this experiment
2. Start Power (Complex?)
3. Start Coherence
4. Start Intensity
5. End Power (Complex)
6. End Coherence
7. End Intensity

We can now try and plot these, such that we can compare them to the original paper.

```
saveFigToDir (generic function with 1 method)
```

```
"Plot values succesfully extracted"
```





Start and end coherence heatmaps for different slit counts

We now have some preliminary results that match our original paper, we can try and explore further. For example, in the FORTRAN code we requested, it seems as though they choose to store the entire matrix A before and after running the experiment. We can do one better, choosing to plot it. In the cell below, we print a set of heat maps that represent the coherence of the light after passing through different slit counts. We did not plot these in this notebook, instead choosing to save them. They can be found in 'figs/heatmaps/'

```
exp_2_args =   GlobalArgs(
                 n = 101
                 zmax = 0.5
                 dd = 0.1
               )
```

```
exp_2_args = GlobalArgs(101, 0.5, 0.1)
```

☑

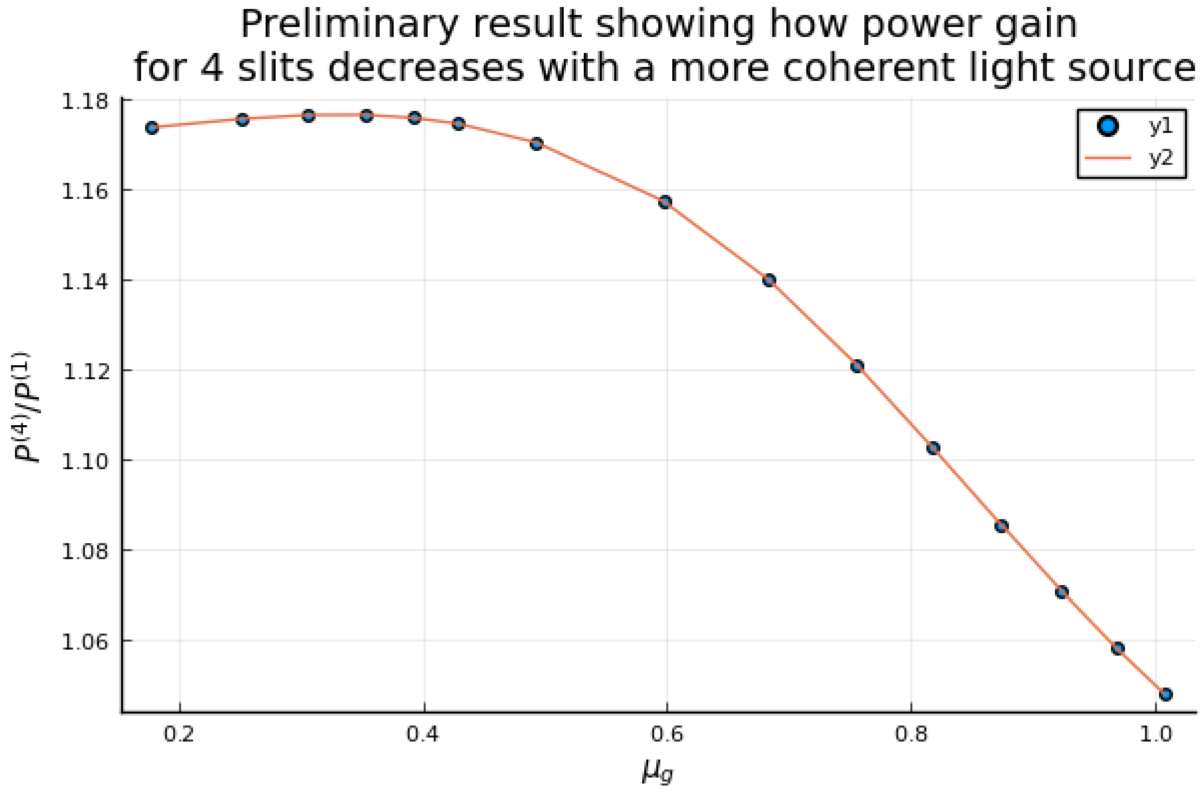
Power gain for a single slit count with respect to different starting coherence

Below, we calculate and plot the way that the power increase ratio for a single slit count $P^{(4)}/P^{(1)}$ for different starting coherences of light.

```
exp_3_args =   GlobalArgs(101,  0.5,  0)
```

☑

```
@bind run_exp_3 CheckBox()
```



We see that this form of the classical zeno effect seems to be mroe prominant for less coherent light sources.

Increasing range of calculation parameters

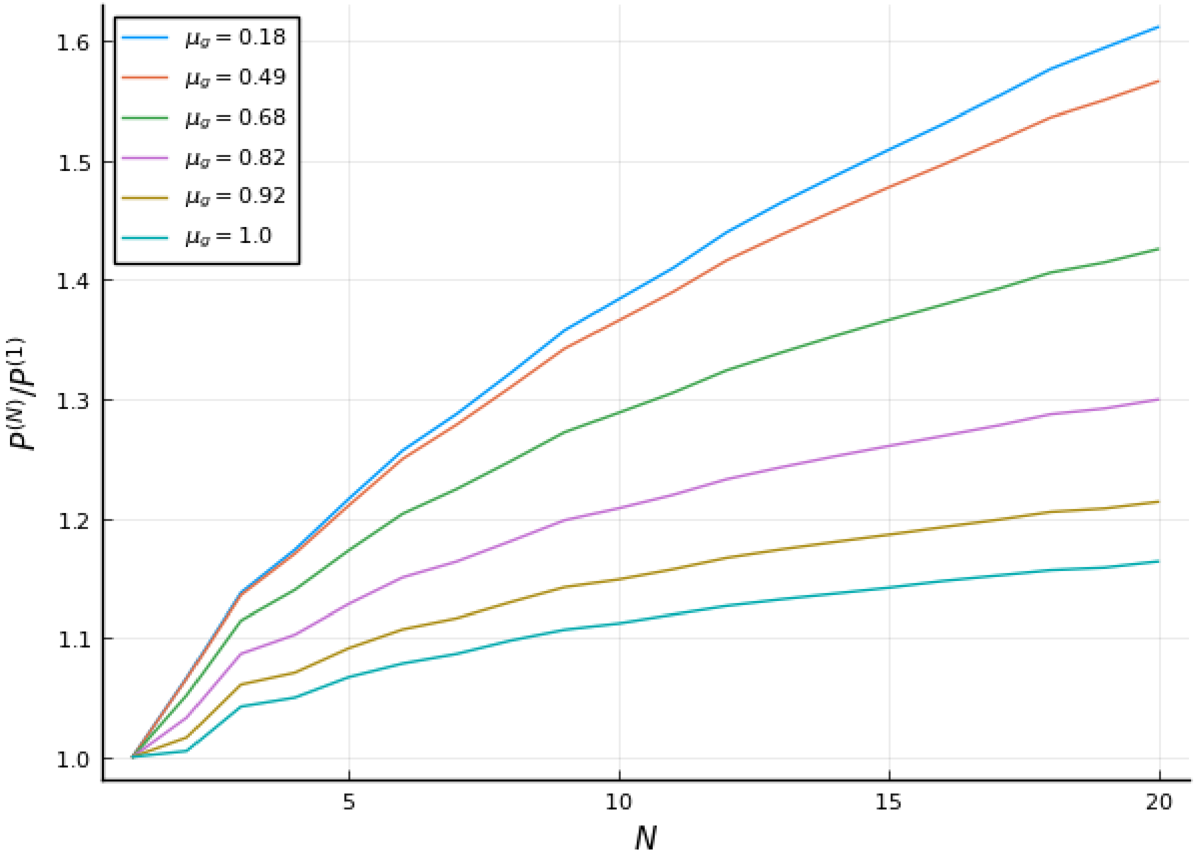
We have now created the main set of graphs we wish to plot, and so we can extend the range of paramters we calculate, to give more informative plots.

```
exp_final_args =  GlobalArgs(101, 0.5, 0)
```

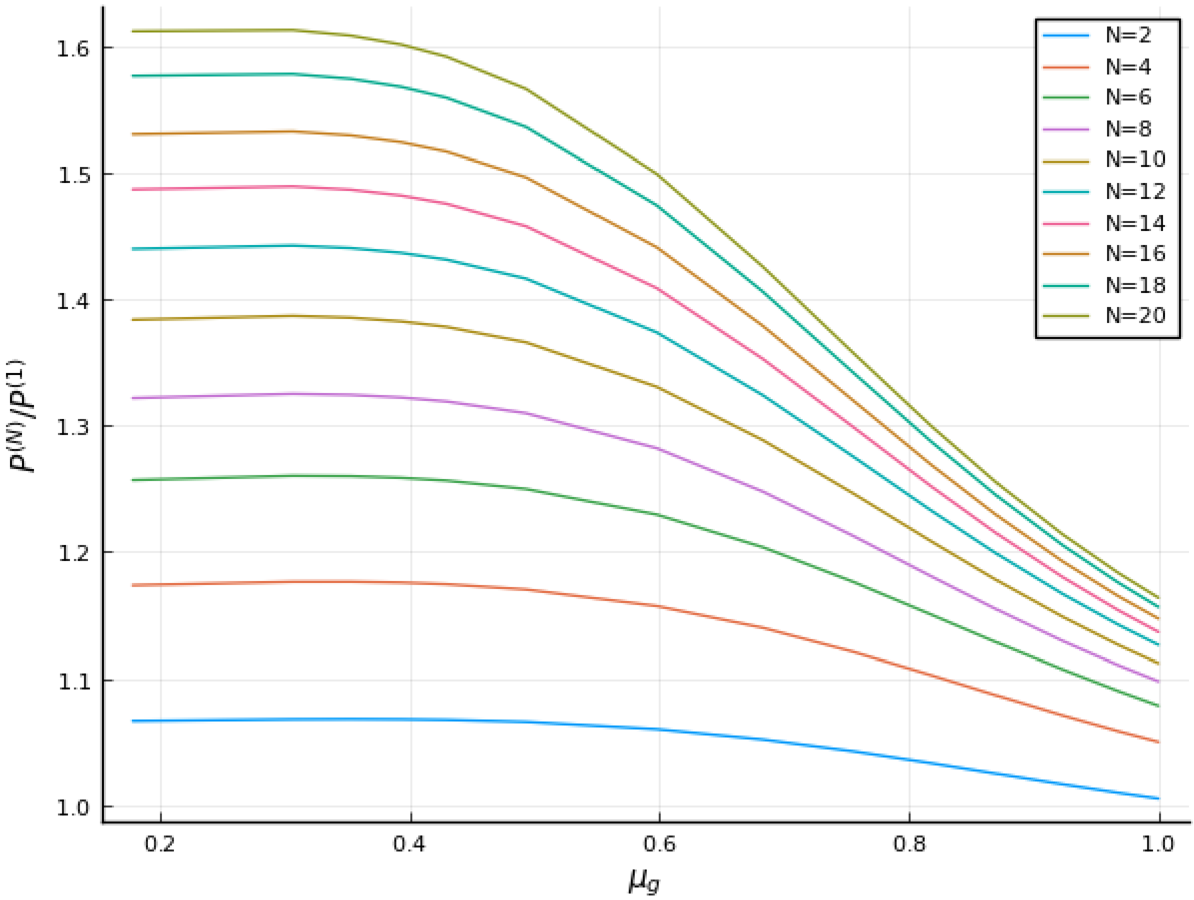
☑

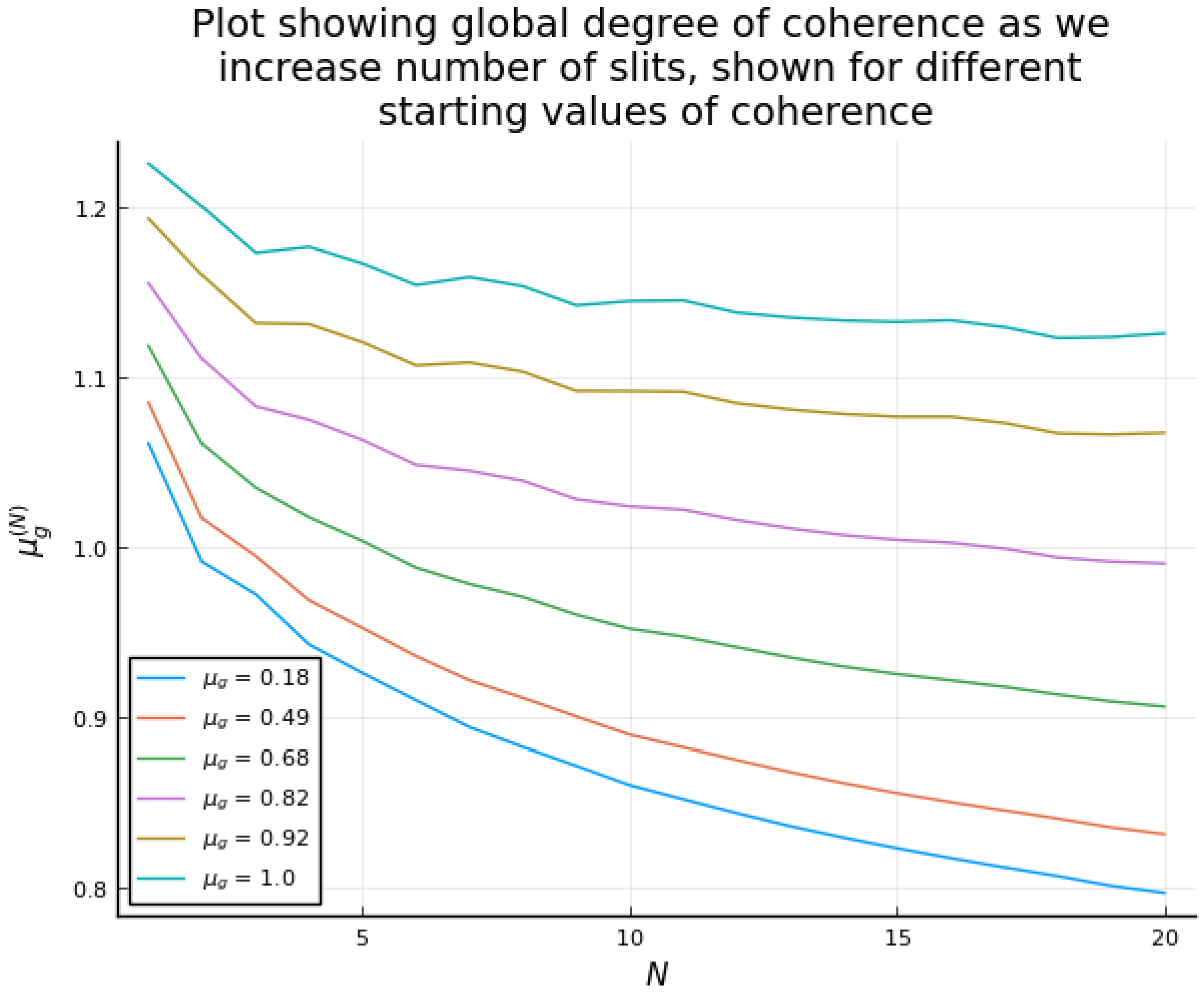
```
@bind run_final CheckBox()
```

Plot showing power gain as a function of interference slit count, shown for multiple starting coherence values



Plot showing power gain for different starting coherence values, for multiple numbers of slits





Jump in coherence and power at each slit

The final aspect of this effect we wish to show, is how the coherence of light changes whenever a slit is encountered. To do this, we must modify our iterate over slits function to allow it to calculate the coherence after each slit. We can then return all these intermitent coherence values to be used for plotting. We shall also return the power at each slit.

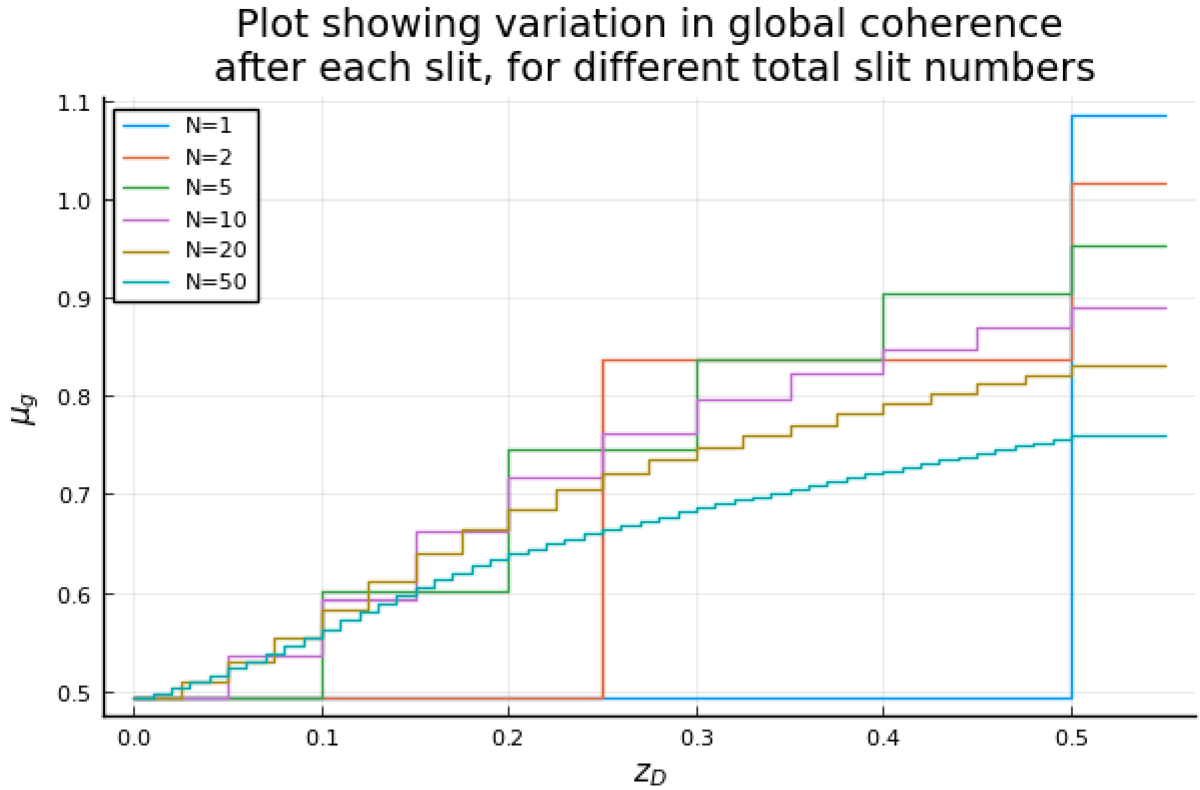
```
Main.workspace3.iterateOverSlitsAndGetCoh
```

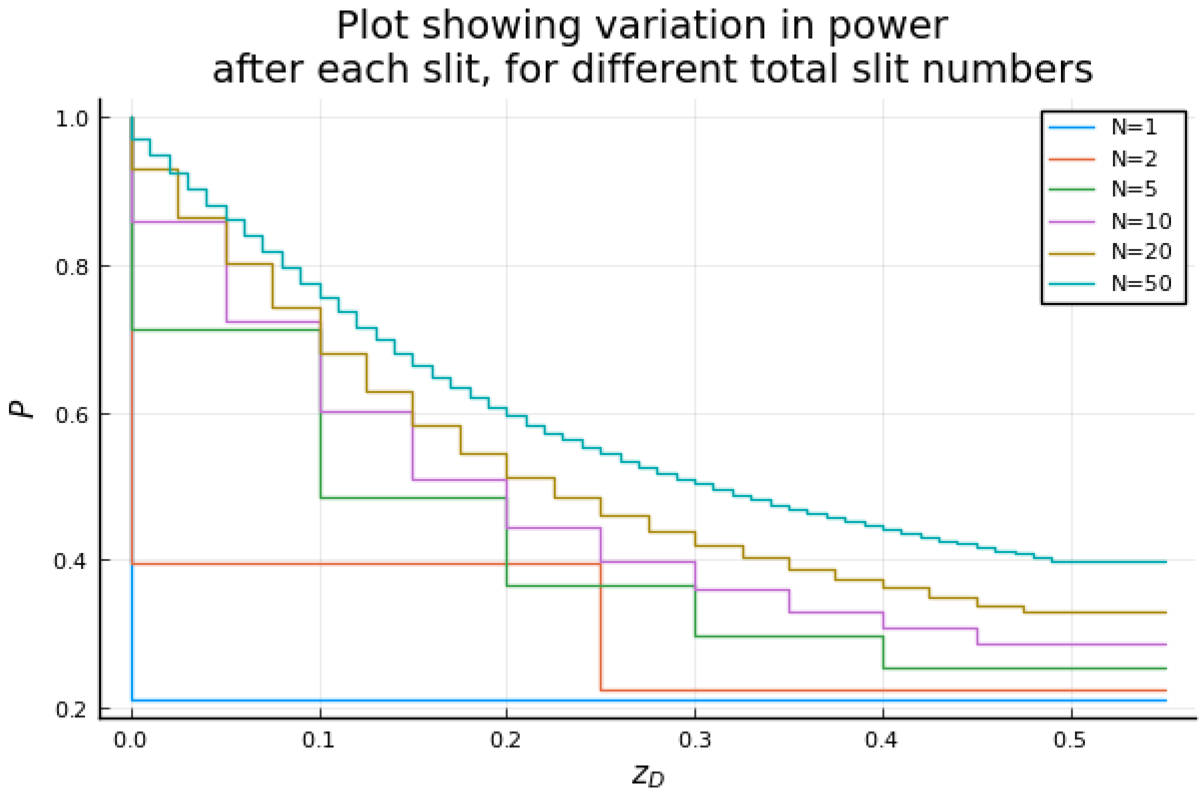
```
exp_jump_args = GlobalArgs(101, 0.5, 0.2)
```

☑

```
"Tick above to run calculation"
```

```
"Coherence jumps measured succesfully"
```


Plot showing variation in global coherence
after each slit, for different total slit numbers
N=1
N=2
N=5
N=10
N=20
N=50
μ_g
z_D
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.0
0.1
0.2
0.3
0.4
0.5


Plot showing variation in power
after each slit, for different total slit numbers
N=1
N=2
N=5
N=10
N=20
N=50
P
z_D
1.0
0.8
0.6
0.4
0.2
0.0
0.1
0.2
0.3
0.4
0.5