Jade Ducharme

Complete Summary

In this paper, I will describe everything I have done so far since the start of the summer. I divided the task into three different Python files: "helper_functions.ipynb", "Scaling_Solution_Signal.ipynb", and "Gaussian_Fluctuations.ipynb".

Helper Functions

The equations used in this file were adapted from a paper titled "A Signature of Cosmic String Wakes in the CMB Polarization" (2010).

This first file contains all helpful constants, conversions, and other functions that will be necessary to construct the signal. Specifically, it contains the following list of constants in natural units:

Table 1: All useful constants in natural and non-natural units

Name	Symbol	Value in non-natural units	Constant of proportionality	Value in natural units
Thomson cross section	σ_T	6.652×10^{-29} m^2	$(\frac{1\text{GeV}^{-1}}{1.973\times10^{-16}\text{m}})^2$	1708.82 GeV^{-2}
Ionization fraction	f	10^{-3}		10^{-3}
String tension	$G\mu$			3×10^{-7}
Baryon fraction	Ω_B	0.2		0.2
Planck mass	m_P			1.221×10^{19} GeV
Inverse proton mass	m_p^{-1}	5.9787×10^{26} kg^{-1}	$\frac{1.7827\times10^{-27}\text{kg}}{1\text{GeV}}$	1.0658 GeV^{-1}
Current time	t_0	$4.361 \times 10^{17} s$	$\frac{1\text{GeV}^{-1}}{6.5823\times10^{-25}\text{s}}$	6.6253×10^{41} GeV^{-1}

The file also contains functions used to retrieve the time at the start of every Hubble time set (computed using $t_{n+1} = 2t_n$, $t_0 = 370000$), the Hubble angular distance at the start of every Hubble time step (computed using $d_c(t_{n+1}) = 2^{1/3} d_c(t_n)$, $d_c(t_0) \simeq 1.8°$), and the pixel size of every Hubble time step relative to an observer on Earth today (computed using a proportionality relation $p(d_c(t_n)) = d_c(t_n) \times \frac{400}{10}$.)

Next, I wrote a CosmicString class and initialized useful cosmic string parameters:

- **Random variable,** c_1: Selected at random in the interval (0.1, 1)
- **Time at which wake is laid down,** t_i: Controlled variable. Selected from elements in the "Hubble time steps" list

- **Redshift at which the wake is laid down,** z_i: Computed using the "z_at_value()" function, which takes in as input a time and returns the redshift corresponding to that time.
- **Time at which the photons cross the wake,** t: Controlled variable. Selected from elements in the "Hubble time steps" list
- **Redshift at which the photons cross the wake,** z**:** Computed using the "z_at_value()" function
- **String length,** l: Computed using $l = c_1 t_i$
- **String velocity,** v_s: Selected at random in the interval (0, 1)
- **Relativistic gamma factor,** γ_s: Computed using $\gamma_s = \frac{1}{\sqrt{1-v_s^2}}$
- **String wake depth,** d: Computed using $d = t_i v_s \gamma_s$
- **Angle relative to observer's line of sight,** θ: Selected at random in the interval (0, $\frac{\pi}{2}$)
- **String wake height,** $h(t, t_i)$: Computed using $h(t, t_i) = \frac{24\pi}{5} G\mu v_s \gamma_s (z_i + 1)^{-1/2} t_0 \frac{z_i+1}{(z+1)^2}$
- **Distance photons travel through the wake,** ℓ: Steps taken to compute this quantity are described below.
- **Polarization signal from a single string wake,** $\frac{P}{Q}$: Computed using $\frac{P}{Q} \simeq \frac{1}{10}(\frac{3}{4\pi})^{1/2} \sigma_T \ell f \rho_B(t_i) m_p^{-1} \frac{(z+1)^4}{(z_i+1)^3}$

Before describing how I arrived at the quantity ℓ, here are a few helpful diagrams:

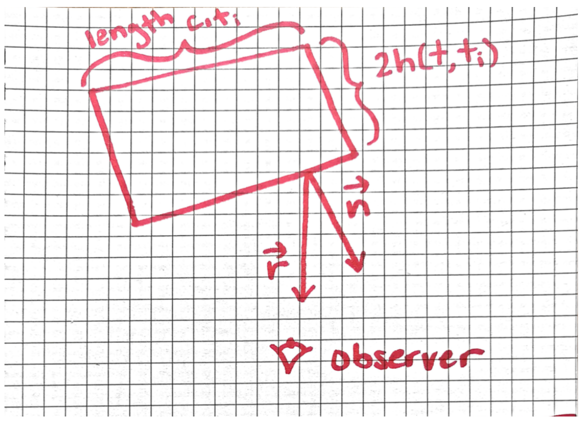


Figure 1: A bird's eye view diagram of a cosmic string wake collapsed in such a way as to hide the wake's depth and present only its length.

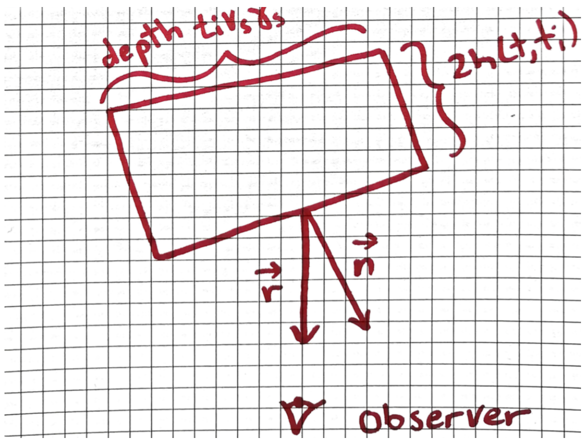


Figure 2: A bird's eye view diagram of a cosmic string wake collapsed in such a way as to hide the wake's length and present only its depth.

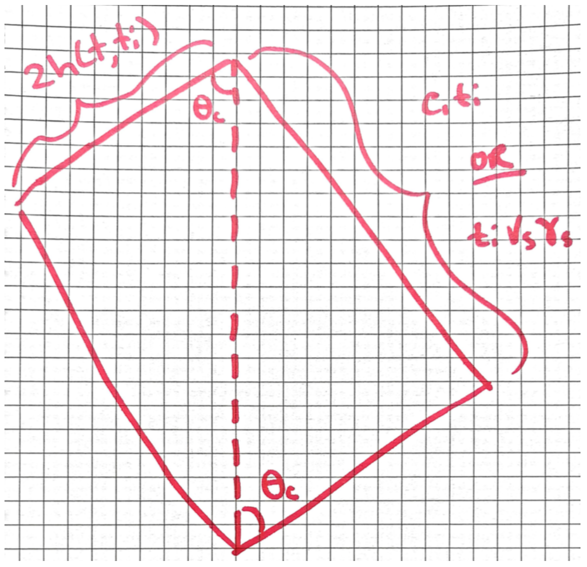


Figure 3: The angle θ_c associated with the longest path the photons can cross through the wake.

Here I will describe how I arrived at the quantity ℓ, the distance photons travel through the wake:

- Assign a random variable (Boolean) to determine whether the wake presents its length l or depth d in a bird's eye view diagram
- Compute the critical angle θ_c using $\theta_c = \tan^{-1}(\frac{l}{2h(t,t_i)})$ or $\theta_c = \tan^{-1}(\frac{d}{2h(t,t_i)})$, as appropriate. The critical angle corresponds to the angle of the wake's diagonal (i.e. the longest path a photon could travel through the wake)
- Compare the critical angle θ_c to the angle θ generated in the CosmicString class.
- If $\theta_c \geq \theta$, compute ℓ using $\ell = \frac{2h(t,t_i)}{\cos(\theta)}$

- If $\theta_c < \theta$, compute ℓ using $\ell = \frac{l}{\sin(\theta)}$ or $\ell = \frac{d}{\sin(\theta)}$, as appropriate

Next, still inside the CosmicString class, I wrote a function called "map_signal" that will retrieve the signal from a single cosmic string and plot it on a 400x400 pixel map. map_signal takes as input the pixel length and pixel depth of the wake. These were found using a proportionality relation. Indeed, let the pixel size of the observable universe at the time of the wake's formation be p (given by the hubble_pixel_size() function described above). Then, the wake's pixel length is $c_1 p$ and its pixel depth is $v_s \gamma_s p$ (rounded to the nearest integer).

Here is a quick diagram to illustrate how the program generates the cosmic strings (descriptive list below):

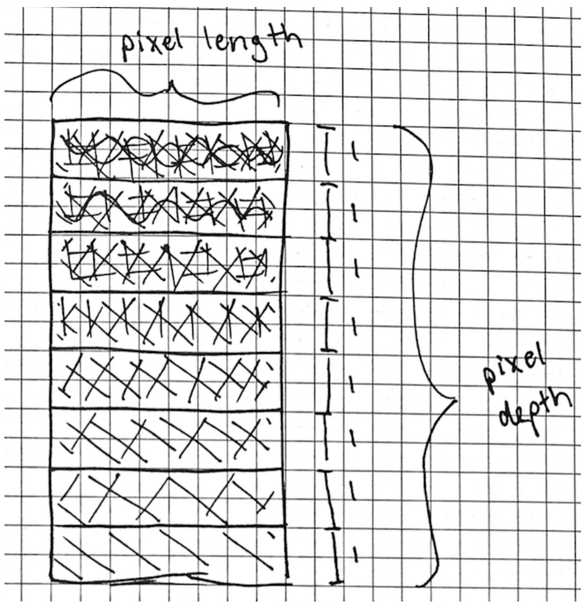


Figure 4: The program draws the wakes piece by piece. Pieces stack, so the polarization signal increases linearly. The first piece laid down by the program is the darkest piece (top), which has the strongest signal. The last piece laid down by the program is the lightest (bottom) with the weakest signal.

Descriptive list:

1. A random variable selected from the range $(0, \pi)$ determines the string's angle relative to the x-axis (tilt)
2. The program draws a rectangle of size length $\times$ 1 (pixels), possessing a signal with a strength of $\frac{2\times\text{polarization\_signal}}{\text{pixel\_depth}}$

3. The program draws a rectangle of size length $\times$ 2 with a signal of the same strength as the previous piece, and adds both pieces
4. The program draws a rectangle of size length $\times$ 3, again with the same signal, and adds all the pieces
5. The program keeps adding pieces until there are a number of pieces corresponding to the pixel depth of the wake. At this point, the polarization signal linearly increases along the depth, going from 0 to twice the average polarization signal
6. Finally, a random variable determines whether the map gets "flipped" or not (using a mirror function), allowing us to reach all possible 360° of rotation

This concludes the Helper Functions file.

Signal from a Scaling Solution of Cosmic Strings

This second file contains the code used to compute the "double sum" in order to account for all the times photons cross wakes.

First, the value N, which is the number of wakes per Hubble volume (a controlled variable), is scaled by the angular size of that Hubble volume. To compute this, we need the total extent of our map (10°) as well as a, the angular size of the Hubble volume of interest. Then, the "scaled" value of N is:

$$N_{scaled} = \frac{10^2}{a^2} \times N$$

N_{scaled} corresponds to the total number of wakes laid down at the time of interest which are captured in the map. For example, for N = 3 and $t_i = t_{rec}$, $N_{scaled} = \frac{10^2}{1.8^2} \times 3 \approx 93$. So our map captures 93 wakes laid down at t_{rec}.

Let's say that, during the first time step, photons cross N_0 wakes. All of these wakes are laid down at $t_0 = t_{rec}$. Then, during the second time step, photons cross N_1 wakes laid down at t_0 and N_1 wakes laid down at t_1. Then, during the third time step, photons cross N_2 wakes laid down at t_0, N_2 wakes laid down at t_1 and N_2 wakes laid down at t_2. And so on until the very last time step.

So, the outer sum is a summation over all the time steps t, while the inner sum is a summation over all $t_i \leq t$. The final graph is a result of this so-called double sum.

This concludes the Scaling Solution Signal file.

Gaussian Fluctuations

This final file produces a map of Gaussian fluctuations. The method used to produce this map was adapted from the one described in a paper titled "Canny Algorithm, Cosmic Strings and the Cosmic Microwave Background" (2010).

First, the relevant constants and equations used throughout the program are listed below:

Constants:

$$N_{max} = 401$$

$$k_{max} = \frac{N_{max}}{2}$$

$$L = 10$$

Equations:

$$k_x = \frac{2\pi}{10}(k_{xn} - k_{max}) \tag{1}$$

$$k_y = \frac{2\pi}{10}(k_{yn} - k_{max}) \tag{2}$$

$$||\mathbf{k}|| = k = \sqrt{k_x^2 + k_y^2} \tag{3}$$

$$B(\mathbf{k}) = \sqrt{\frac{C_l}{2}}(g_1 + ig_2) \tag{4}$$

where k_{xn} and k_{yn} are integers ranging between 0 and N_{max}, and $k_{xn} = k_{yn}$ for a given n value (n ranges from 1 to N_{max} and represents rows in the case of k_x and columns in the case of k_y). g_1 and g_2 are random variables selected from a normal distribution with mean 0 and variance 1.

Next, I produce 1D arrays of k_x and k_y values using Equation 1. From these 1D arrays, I construct a 2D "intersection" array of k values using Equation 3. This $N_{max} \times N_{max}$ array represents all the possible combinations of k_x and k_y values.

Next, I have defined an $l(k)$ function in order to retrieve the spherical harmonics component l associated with each element in the k-array.

$$l(k) = \frac{360k}{2\pi}$$

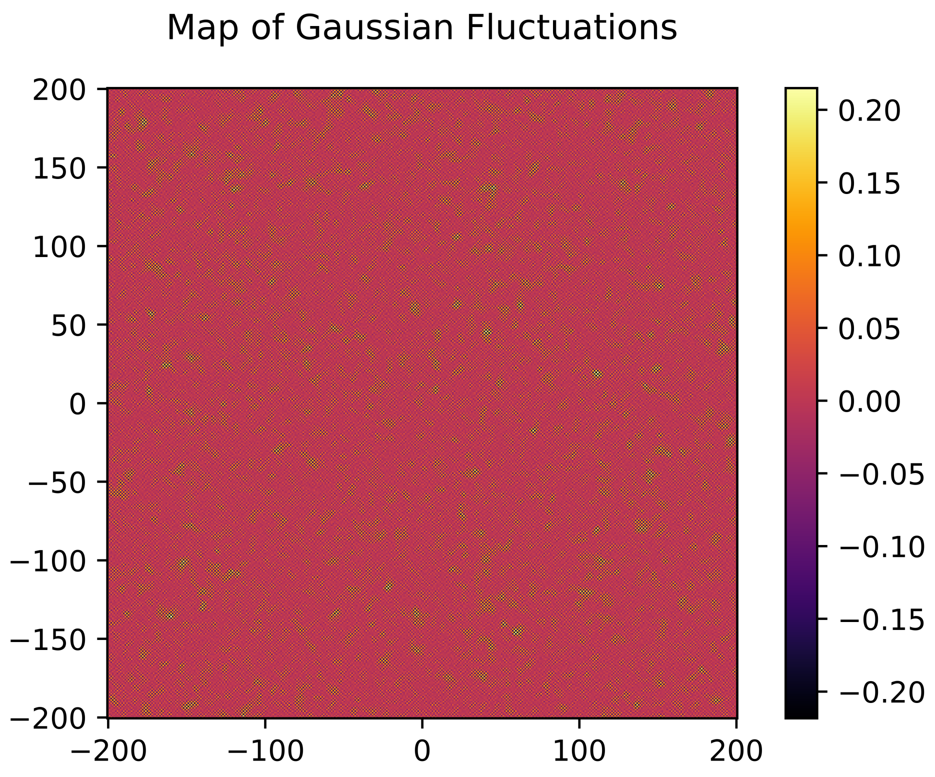
I pass the k-array through the $l(k)$ function to obtain the l-array. Every element in the l-array in rounded to the nearest integer.

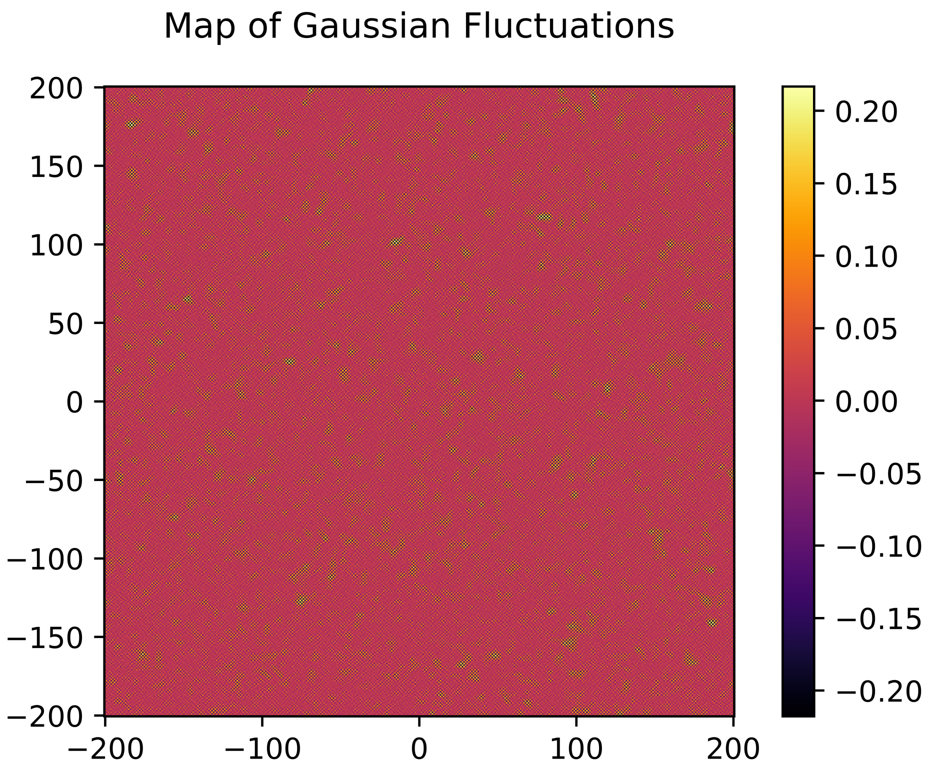
Next, from the CAMB code, I retrieve an array containing every possible C_l for values of l ranging from 1 to an arbitrarily high number (I have set the upper limit to $l = 11000$, since I doubt we will ever need higher l values than that). From there, I relate every element in the l-array to the corresponding C_l value, obtaining a C_l-array.

Next, I construct a Fourier space array by passing the C_l-array through Equation 4. The g_1 and g_2 random variables have a different iteration for every element in the array (as it should be).

Finally, I retrieve the position space map by passing the Fourier space array through the numpy.fft.fft2() function. This is a built-in function in Python that performs a 2D Fast Fourier Transform on a given 2D array. At that point, we are left with a 401 × 401 array. But, in order to compare with the cosmic string maps, we need a 400 × 400 array. So, I simply delete the first row and column of the position space map. Then, I take the real part and plot it.

This concludes the Gaussian Fluctuations file. Here are two example plots generated in the Gaussian Fluctuations file:





Map of Gaussian Fluctuations
200
150
100
50
0
−50
−100
−150
−200
−200
−100
0
100
200
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20