**Generation of Random Variables:**

The generation of random variables is a fundamental concept in statistics and computer science. Random variables are used to model uncertainty and randomness in various applications, such as simulations, statistical experiments, and probabilistic modeling.

**There are two main methods for generating random variables:**

**1. True Random Number Generation (TRNG):**

- True random number generators generate random values from a source of true randomness, often based on physical processes like electronic noise or radioactive decay.

- TRNGs are considered truly random because the outcomes are unpredictable and not determined by any algorithm.

- Examples of TRNG sources include radioactive decay, atmospheric noise, and quantum processes.

- True randomness is essential for cryptographic applications and security.

TRNGs are devices that generate numbers from a physical process. These processes are inherently unpredictable and include phenomena like electronic noise, nuclear decay, or chaotic systems. For simulations, especially in cryptography and where security is critical, TRNGs are crucial because they provide non-deterministic, unpredictable outputs.

Here are a few ways to get true random numbers for your simulations:

1. **Hardware TRNGs:** These are physical devices designed to generate true random numbers, often used in security-sensitive contexts. They can be standalone devices or integrated circuits within a larger system.
2. **Online Services:** Websites like RANDOM.ORG provide true random numbers generated from **atmospheric noise.** They allow you to download sequences of random numbers for your simulations.

People use RANDOM.ORG for holding drawings, lotteries, and sweepstakes, to drive online games, for scientific applications, and for art and music. The service has existed since 1998 and was built by [Dr Mads Haahr](https://www.random.org/mads/) of the [School of Computer Science and Statistics](http://www.scss.tcd.ie/) at [Trinity College, Dublin](http://www.tcd.ie/) in Ireland. Today, RANDOM.ORG is operated by [Randomness and Integrity Services Ltd](https://www.random.org/company/).

1. **Software Libraries:** Some programming languages and libraries can interface with hardware TRNGs in your computer or access online services to provide true random numbers in your applications.
2. **Quantum Random Number Generators:** These are based on quantum phenomena such as quantum noise and are typically available through specialized hardware or online services.

For simulations that do not require the high level of unpredictability provided by TRNGs, pseudorandom Number Generators (PRNGs) are often sufficient and more easily accessible through standard programming libraries.

**2. Pseudo-Random Number Generation (PRNG):**

- Pseudo-random number generators are algorithms that produce sequences of numbers that appear to be random but are generated using **deterministic algorithms.**

- PRNGs start with an initial value called **a seed** and then use mathematical formulas to produce a sequence of numbers.

- While PRNGs are not truly random, they are often sufficient for many applications where a high degree of randomness is not required.

- PRNGs are widely used in simulations, computer games, statistical sampling, and scientific computing.

**\*Pseudo-random Number Generators (PRNGs):**

Pseudo-random number generators are commonly used for generating random variables in computer programs and simulations. They are deterministic algorithms that produce sequences of numbers that mimic the statistical properties of true random sequences.

**Some key characteristics of PRNGs include:**

- Seed: PRNGs require an initial value called a seed. The same seed will produce the same sequence of pseudo-random numbers. Changing the seed can lead to different sequences.

- Periodicity: PRNGs have a finite period, meaning that the sequence of numbers will eventually repeat itself. The length of this period depends on the specific PRNG algorithm.

- Reproducibility: PRNGs are reproducible, meaning that if you start with the same seed and use the same algorithm, you will get the same sequence of pseudo-random numbers. This property is essential for debugging and replicating experiments.

Common PRNG algorithms include the **Linear Congruential Generator (LCG**), Mersenne Twister, and Xorshift. These algorithms balance the trade-off between computational efficiency and the quality of randomness.

In summary, the generation of random variables is a crucial component of various fields, including statistics and computer science. While true random number generators provide genuine randomness, pseudo-random number generators are widely used for their efficiency and reproducibility in applications where true randomness is not strictly required. PRNGs are especially valuable for simulations, modeling, and algorithm development.

* 1. **Properties of a Good Random Number Generator** The random numbers generated should;
     1. Have as nearly as possible a uniform distribution
     2. Should be fast
     3. Not require large amounts of memory
     4. Have a long period
     5. Be Able to generate a different set of random numbers or a series of numbers
     6. Not degenerate.

A **linear congruential generator** (**LCG**) is an algorithm that yields a sequence of

**pseudo-randomized numbers** calculated with a discontinuous piecewise linear equation. The method represents one of the oldest and best-known pseudorandom number generator algorithms. The theory behind them is relatively easy to understand, and they are easily implemented and fast, especially on computer hardware which can provide modulo arithmetic by storage-bit truncation.

The generator is defined by the recurrence relation:





where is the sequence of pseudorandom values, and



* the "modulus"



* + the "multiplier"



* + the "increment"



– the "seed" or "start value"

# Example 1

For example, the sequence obtained when X0 = a = c = 7, m = 10, is 7, 6, 9, 0, 7, 6, 9, 0, ...

And suppose m = 8, a = 5, c = 7 and X0 (seed) = 4 we can generate a random sequence of integer numbers thus:

# 

And suppose m = 8, a = 5, c = 7 and X0 (seed) = 4 we can generate a random sequence of integer numbers thus:

|  |  |  |
| --- | --- | --- |
| X1-seed |  | 4 |
| X2 | (5\*4 +7) mod 8 =27 mod 8 | 3 |
| X3 | (5\*3 +7)mod 8 =22 mod 8 | 6 |
| X4 | (5\*6 + 7) mod 8 = 37 mod 8 | 5 |
| X5 | (5\*5 + 7) mod 8 = 37 mod 8 | 0 |
| X6 | (5\*0 + 7) mod 8 = 7 mod 8 | 7 |
| X7 | (5\*7+ 7) mod 8 = 42 mod 8 = 5 | 2 |
| X8 | (5\*2+ 7) mod 8 = 17 mod 8 = 1 | 1 |
| X9 | (5\*1+ 7) mod 8 = 12 mod 8 = 4 | 4 |

Random numbers are **3,6,5,0,7,2,1,4**

# Table 2: Random sequence of integer numbers

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | X 1 | = | (5\*X0+7)mod 8  = | | (5\*4+7)mod 8  = | 27 mod 8 = 3 | | | | |
| 1 | X 2 | = | (5\*X1+7)mod 8  = | | (5\*X3+7)mod 8  = | 22 mod 8 = 6 | | | | |
| 2 | X 3 | = | (5\*X2+7)mod 8  = | | (5\*X6+7)mod 8  = | 37 mod 8 = 5 | | | | |
| 3 | X 4 | = | (5\*X3+7)mod 8  = | | (5\*X5+7)mod 8  = | 32 mod 8 = 0 | | | | |
| 4 | X 5 | = | (5\*X4+7)mod 8  = | | (5\*X0+7)mo d 8 | | = | 7 mod  8  = 7 | |
| 5 | X 6 | = | (5\*X5+7)mod 8 | = | (5\*X7+7)mo d 8 | | = | 42 mod  8 | = 2 |
| 6 | X 7 | = | (5\*X6+7)mod 8 | = | (5\*X2+7)mo d 8 | | = | 17 mod  8 | = 1 |
| 7 | X 8 | = | (5\*X7+7)mod 8 | = | (5\*X1+7)mo d 8 | | = | 12 mod  8 | = 4 |

**TEST OF RANDOMNESS**

1. **Chi-Squared Test for Uniformity**:

**Scenario:** You have generated 1000 random numbers between 1 and 10 (inclusive) using a random number generator, and you want to test if these numbers are uniformly distributed across the 10 possible values.

**Step 1: Define Hypotheses:**

* Null Hypothesis (H0): The generated numbers follow a uniform distribution.
* Alternative Hypothesis (Ha): The generated numbers do not follow a uniform distribution.

**Step 2: Set up Intervals:**

* Divide the range of possible values (1 to 10) into intervals. In this case, each interval corresponds to a single number.
* You'll have 10 intervals: [1], [2], [3], [4], [5], [6], [7], [8], [9], [10].

**Step 3: Count Observed Frequencies:**

* Count how many of the 1000 generated numbers fall into each interval.

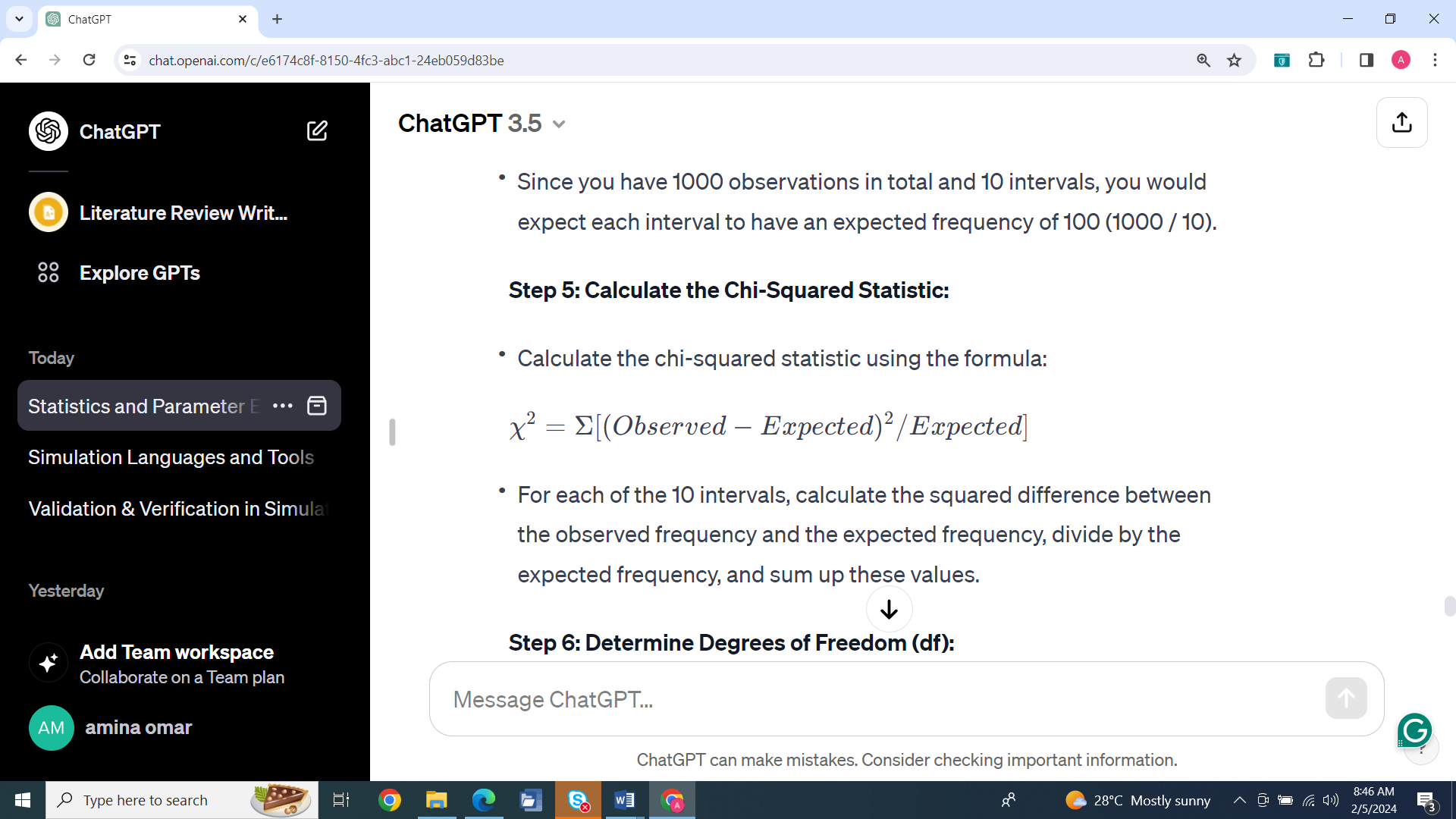
Here's an example table of observed frequencies:

| **Interval** | **Observed Frequency** |
| --- | --- |
| [1] | 102 |
| [2] | 97 |
| [3] | 98 |
| [4] | 104 |
| [5] | 94 |
| [6] | 98 |
| [7] | 103 |
| [8] | 101 |
| [9] | 99 |
| [10] | 104 |

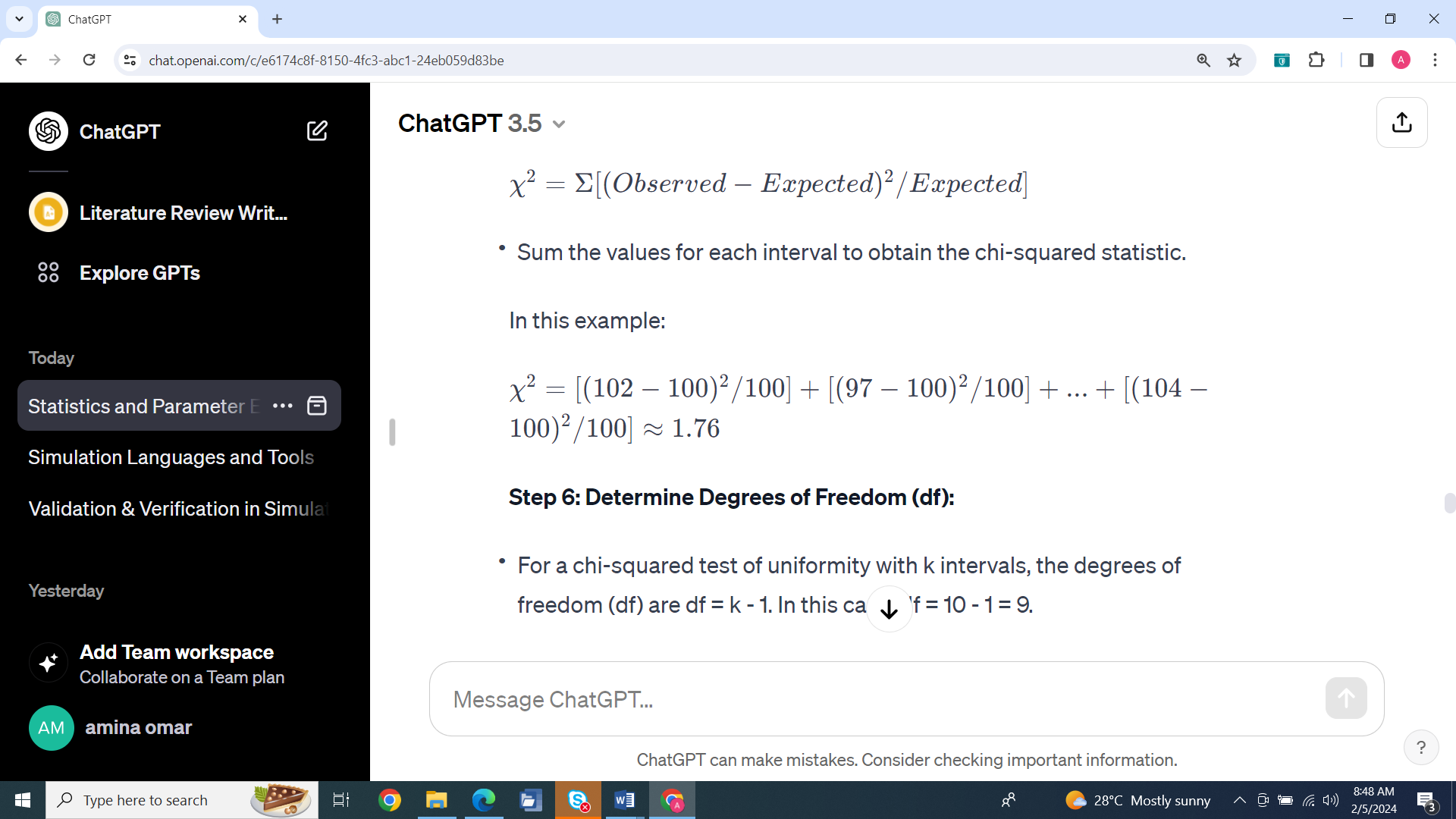
**Step 4: Calculate Expected Frequencies:**

* Assuming a uniform distribution, you would expect each interval to have roughly the same number of observations.
* Since you have 1000 observations and 10 intervals, the expected frequency for each interval would be 1000 / 10 = 100.

**Step 5: Calculate the Chi-Squared Statistic:**

* Calculate the chi-squared statistic using the formula:
* 
* Sum the values for each interval to obtain the chi-squared statistic.

In this example:



**Step 6: Determine Degrees of Freedom (df):**

* For a chi-squared test of uniformity with k intervals, the degrees of freedom (df) are df = k - 1. In this case, df = 10 - 1 = 9.

**Step 7: Choose Significance Level (α):**

* Typically, a significance level of 0.05 is used.

**Step 8: Find the Critical Chi-Squared Value:**

* You can use a chi-squared distribution table or calculator to find the critical chi-squared value corresponding to α and df. For α = 0.05 and df = 9, the critical value might be approximately 16.92.

**Option 1: Using a Chi-Squared Distribution Table:**

* Chi-squared distribution tables provide critical values for various levels of significance (α) and degrees of freedom (df).
* Locate the row in the chi-squared distribution table corresponding to your chosen significance level (α) and the column corresponding to your degrees of freedom (df). The intersection of that row and column will give you the critical chi-squared value.

**Option 2: Using a Calculator or Software:**

* Most statistical software packages and scientific calculators can calculate critical chi-squared values. You can use functions like **chi2inv** in Excel, **qchisq** in R, or chi-squared calculators available online. =chiinv (probability, degree of freedom)
* Simply input your chosen significance level (α) and degrees of freedom (df) into the calculator or software, and it will provide you with the critical chi-squared value.

In your specific example, you have α = 0.05 (a common significance level) and df = 9 (because you have 10 intervals, and degrees of freedom are df = 10 - 1 = 9).

When you look up or calculate the critical chi-squared value for α = 0.05 and df = 9, you will find that it is approximately **16.92.** This means that in your chi-squared test, if the calculated chi-squared statistic (based on your observed and expected frequencies) is less than 16.92, you will fail to reject the null hypothesis. If it's equal to or greater than 16.92, you will reject the null hypothesis. This critical value helps you make decisions about the statistical significance of your test results.

**Step 9: Compare Calculated Chi-Squared Value to Critical Value:**

* If χ² < critical value, fail to reject the null hypothesis (H0), indicating that the data is consistent with a uniform distribution.
* If χ² ≥ critical value, reject the null hypothesis (H0), suggesting that the data does not follow a uniform distribution.

In this example, χ² = 1.76, which is less than the critical value of 16.92. Therefore, you would fail to reject the null hypothesis and conclude that the generated numbers are consistent with a uniform distribution.