Homework 5 - BIOMATH 205

SIMON LEE

November 2023

1 Chapter 11

Q1: Write and test a fast julia function to multiply two large positive integers?

This Julia code demonstrates a fast multiplication of two large positive integers using the Fast Fourier Transform (FFT) algorithm (Schönhage–Strassen algorithm) The multiply_fft function pads the input vectors to the next power of 2 for efficient FFT (refer to Cooley-Tukey FFT if you are interested why power of 2's result in fast FFT), performs FFT on the padded vectors, multiplies them pointwise in the frequency domain, and then applies the inverse FFT to obtain the result in the time domain. The real part of the result is rounded to integers. The nextpow2 function finds the next power of 2 greater than or equal to a given number. We utilized the FFTW package in Julia for FFT calculations.

We demonstrate in the code below a normal multiplication with small values and the fast multiplication with large positive integers and see a noticeable difference both in memory consumption and execution time.

```
import Pkg
Pkg.add("FFTW")
using FFTW

function multiply_fft(a, b)
    # Pad input vectors to the next power of 2 for efficient FFT
    n = nextpow2(length(a) + length(b) - 1)
    a_padded = vcat(a, zeros(eltype(a), n - length(a)))
    b_padded = vcat(b, zeros(eltype(b), n - length(b)))

    # Perform FFT on padded input vectors
    fft_a = fft(a_padded)
    fft_b = fft(b_padded)

    # Pointwise multiplication in frequency domain
    fft_result = fft_a .* fft_b
```

```
    # Inverse FFT to get the result in time domain
    result = ifft(fft_result)

    # Round real part to integers
    result_real = round.(Int, real(result))

    return result_real
end

function nextpow2(n)
    # Find the next power of 2 greater than or equal to n
    power = 0
    while 2^power < n
        power += 1
    end
    return 2^power
end


a = 123
b = 456
c = 123456789
d = 987654321


println("Timing the multiplication of a and b:")
@time result = multiply_fft(a, b)

println("Result of multiplication: ", result)

println("Timing the multiplication of c and d:")

@time result2 = multiply_fft(c, d)

println("Result of fast multiplication on large numbers: ", result2)

>>> Timing the multiplication of a and b:
  0.053173 seconds (52.84 k allocations: 2.641 MiB, 98.66% compilation time)
Result of multiplication: [56088]
Timing the multiplication of c and d:
  0.000077 seconds (24 allocations: 1.734 KiB)
Result of fast multiplication on large numbers: [121932631112635264]
```

Q18: Show that the transpose of a circulant matrix is circulant. Under what condition is a circulant matrix symmetric. What if we replace transpose with adjoint (conjugate transpose) and symmetric by Hermitian (self-adjoint).

A circulant matrix is a square matrix where each row is a right cyclic shift of the previous row. Consider a general $n \times n$ circulant matrix A. A circulant matrix can be represented as follows:

$$A = \begin{pmatrix} a_0 & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_1 & a_0 & a_{n-1} & \dots & a_2 \\ a_2 & a_1 & a_0 & \dots & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 \end{pmatrix} \tag{1}$$

We can see that if you do a right cyclic shift on the first row you get the second row, and this pattern continues.

Transpose of a Circulant Matrix

Now, let's find the transpose of matrix A:

$$A^T = \begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{n-1} \\ a_{n-1} & a_0 & a_1 & \dots & a_{n-2} \\ a_{n-2} & a_{n-1} & a_0 & \dots & a_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \dots & a_0 \end{pmatrix} \tag{2}$$

The transpose of a matrix A, denoted A^T, involves flipping the matrix over its diagonal. As you can see, the transpose of A is also a circulant matrix, since each row is a cyclic permutation of the previous row.

Symmetry of a Circulant Matrix

Now, let's discuss under what condition a circulant matrix is symmetric. If we show that $A = A^T$ then it is equivalent to saying it is symmetric. Therefore to get a symmetric matrix the condition $a_i = a_{n-i}$ for all i where a_i are the elements of the matrix. This means that the entries are symmetric about the center of the matrix

Adjoint and Hermitian of a Circulant Matrix

Now we replace transpose with adjoint and symmetric by Hermitian. A matrix A is hermitian if its conjugate transpose equals itself $A = A^H$ Therefore in order to get a Hermitian matrix you need to do the take the transpose of the matrix and then take the complex conjugate of each entry:

$$A = \begin{pmatrix} a_0 & a_{n-1} & a_{n-2} & \dots & a_1 \\ a_1 & a_0 & a_{n-1} & \dots & a_2 \\ a_2 & a_1 & a_0 & \dots & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \dots & a_0 \end{pmatrix} \tag{3}$$

$$A^H = \begin{pmatrix} \overline{a_0} & \overline{a_1} & \overline{a_2} & \dots & \overline{a_{n-1}} \\ \overline{a_{n-1}} & \overline{a_0} & \overline{a_1} & \dots & \overline{a_{n-2}} \\ \overline{a_{n-2}} & \overline{a_{n-1}} & \overline{a_0} & \dots & \overline{a_{n-3}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \overline{a_1} & \overline{a_2} & \overline{a_3} & \dots & \overline{a_0} \end{pmatrix} \tag{4}$$

First we can see that a conjugate transpose matrix is still circulant since each row is still a right cyclic shift of the previous row.

In order to obtain a Hermitian matrix, the values of a_i and a_{n-i} are crucial. Specifically, for the matrix to be Hermitian, this requires that $a_i = \overline{a_{n-i}}$ for all i and $a_{n-i} = \overline{a_i}$. Additionally, the diagonal elements must be real, as they need to be equal to their own complex conjugates, which implies that $a_0 = \overline{a_0}$

Q20: Prove that the sum or product of two circulant matrices of the same size is circulant. Also prove that the inverse of an invertible circulant matrix C is circulant. Thus, the collection of invertible circulant matrices forms a group. Why is this group commutative?

A circulant matrix is a square matrix where each row is a right cyclic shift of the previous row. Let's consider two circulant matrices A and B of the same size $3x3$. The elements of these matrices can be represented as follows:

$$A = \begin{pmatrix} a_{00} & a_{10} & a_{20} \\ a_{01} & a_{11} & a_{21} \\ a_{21} & a_{12} & a_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{00} & b_{10} & b_{20} \\ b_{01} & b_{11} & b_{21} \\ b_{21} & b_{12} & b_{22} \end{pmatrix}$$

Proof for Sum: The sum of circulant matrices $A + B$ results in each element $a_{ij} + b_{ij}$. Since both A and B are circulant, the sum $A + B$ will also have the property that each row vector is a rotation of the previous row vector. Therefore, $A + B$ is circulant. This scales even at a nxn matrix size.

Product of Two Circulant Matrices

The product of circulant matrices A and B is a matrix D where each element d_{ij} is the dot product of the ith row of A and the jth column of B.

$$D = AB$$

To prove that D is circulant, consider the $(i+1)$th row of A and how it multiplies with columns of B. This row is a right shift of the ith row of A. Since B is circulant, each of its columns is also a right shift of the previous column. The dot product of a right-shifted row with right-shifted columns will yield elements that maintain the circulant property (right shift of previous elements). Thus, the product D is circulant because each row of D is formed by multiplying a right-shifted row of A with appropriately shifted columns of B, preserving the circulant nature. This again scales even at a nxn matrix size.

Inverse of a Circulant Matrix Let C be an nxn invertible circulant matrix. The elements of C can be represented as follows where each row is a cyclic shift of the previous row:

$$C = \begin{pmatrix} c_0 & c_1 & c_2 \\ c_2 & c_0 & c_1 \\ c_1 & c_2 & c_0 \end{pmatrix}$$

We need to show that C^{-1} is also a circulant matrix and can do it with the following:

1. **Diagonalization of Circulant Matrices:** Circulant matrices can be diagonalized by the discrete Fourier transform (DFT) matrix F and its inverse F^{-1}. The matrix F consists of eigenvectors of C, and it is a constant matrix that depends only on the size of C, not on its entries. The diagonalization can be written as:

$$C = FDF^{-1}$$

Here, D is a diagonal matrix containing the eigenvalues of C.

2. **Inverse of the Diagonalized Matrix**: The inverse of C, denoted as C^{-1}, can be found by inverting the diagonalization:

$$C^{-1} = (FDF^{-1})^{-1} = FD^{-1}F^{-1}$$

Since D is diagonal, D^{-1} is simply the inverse of each diagonal element of D.

3. **Preservation of Circulant Structure**: The key point is that the product of F^{-1}, D^{-1}, and F preserves the circulant structure. The matrix F and its inverse are constant for a given size and do not depend on the specific elements of C. The matrix D^{-1}, being diagonal, scales the columns of F but does not alter their circulant arrangement. When F^{-1} is then multiplied by this scaled version of F, the result is still a circulant matrix.

4. **Every Row is a Cyclic Shift**: Since the DFT matrix F and its inverse are structured such that their multiplication with a diagonal matrix results in a matrix where each row is a cyclic shift of the other, the inverse C^{-1} will also have this property, proving that it is circulant. This again scales even at a nxn

matrix size.

Group of Invertible Circulant Matrices is Commutative: A group is a set of elements combined with an operation that satisfies four conditions: closure, associativity, the existence of an identity element, and the existence of inverse elements. We therefore demonstrate these four conditions and why it forms a group:

1. **Closure**: The product of two circulant matrices is circulant.

   If A, B are circulant, then AB is also circulant.

2. **Associativity**: Matrix multiplication is associative.

   For any matrices $A, B, C,$ we have $(AB)C = A(BC)$.

3. **Identity Element**: The identity matrix, which is circulant, serves as the identity element.

   For any circulant matrix $A,$ we have $AI = IA = A,$ where I is the identity matrix.

4. **Invertibility**: Every invertible matrix has an inverse, which is also circulant in this case.

   If A is an invertible circulant matrix, then there exists A^{-1} such that $AA^{-1} = A^{-1}A = I$.

The group is commutative because the multiplication of circulant matrices is commutative. This means $AB = BA$ for any two circulant matrices A and B. This property arises from the fact that circulant matrices are diagonalized by the same unitary matrix, and the multiplication of diagonal matrices is commutative.

In summary, the set of invertible circulant matrices forms a group under matrix addition and multiplication, and this group is commutative because the multiplication of any two circulant matrices commutes.

2 Chapter 12

Q5: Describe one method for generating independent Poisson deviates and implement it in code,

One common method to generate independent Poisson deviates is the poisson process. The Poisson process method, also known as the exponential inter-arrival time method, is a way to generate random samples from a Poisson distribution using the properties of the Poisson process and exponential random variables. The key idea is that in a Poisson process, the times between successive

events (inter-arrival times) are exponentially distributed. From this definition it is designed to generate independent Poisson deviates. The independence comes from the fact that the method relies on generating independent exponential random variables, and the sum of these variables is used to determine the number of events in a fixed time interval.

```
using Random
using Plots

function generate_poisson(lambda, n)
    poisson_count = 0
    cumulative_sum = 0.0

    for _ in 1:n
        exponential_var = -log(rand())
        cumulative_sum += exponential_var

        if cumulative_sum > 1.0
            break
        else
            poisson_count += 1
        end
    end

    return poisson_count
end

seed_value = 123
Random.seed!(seed_value)
lambda = 2.0
num_samples = 100

poisson_samples = [generate_poisson(lambda, 10) for _ in 1:num_samples]

# Plot the histogram of the Poisson samples
histogram(poisson_samples, bins=:auto, xlabel="Poisson Count", ylabel="Frequency", title="Po
```

Refer to **figure 1**
Q8: Based on random normal deviates, discuss how one can generate random deviates from the log normal, chi-squared, F and students t distribution.

Log-Normal Distributions: To generate random deviates from a log-normal distribution, one can take the exponential of a normally distributed random variable. If X is normally distributed with the means μ and standard deviation σ, then $Y = e^X$ follows a log-normal distribution with the parameters μ and σ.

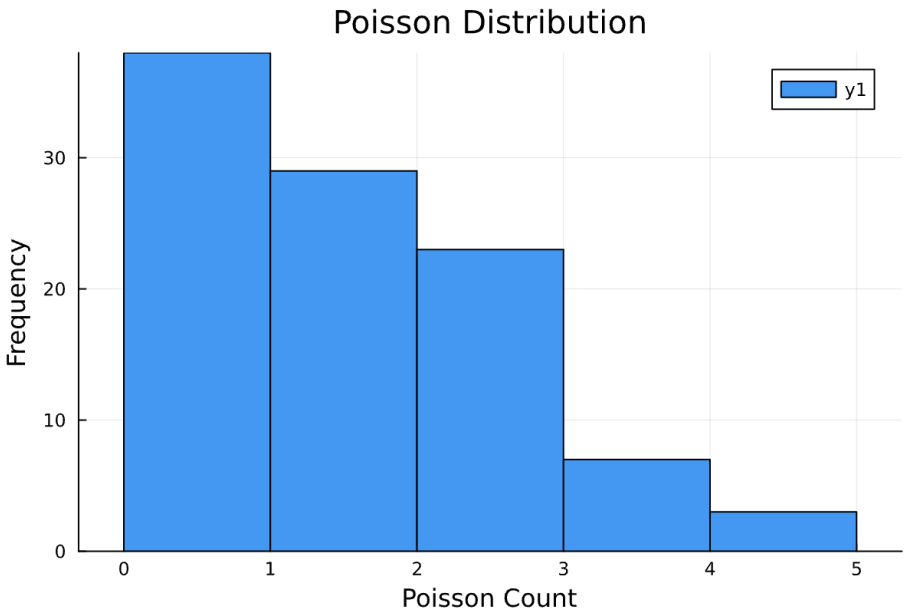

Figure 1: Histogram of our independent poisson deviates that follows a poisson distribution

Chi-squared distributions: For generating random deviates from a chi-squared distribution with k degrees of freedom, sum the squares of k independent standard normal deviates. If $Z_1, Z_2, ..., Z_k$ are independent standard normal variables, then $X = Z_1^2 + Z_2^2 + ... + Z_k^2$ follows a chi-squared distribution with k degrees of freedom.

F-Distributions To generate random deviates from an F-distribution with parameters d_1 and d_2, one can take the ratio of two independent chi-squared variables. If X_1 follows a chi-squared distribution with d_1 degrees of freedom and X_2 follows a chi-squared distribution with d_2 degrees of freedom, then $Y = \frac{X_1/d_1}{X_2/d_2}$ follows an F-distribution with d_1 and d_2 degrees of freedom.

Student's t-Distribution Random deviates from a t-distribution with n degrees of freedom can be generated by dividing a standard normal deviate by the square root of an independent chi-squared variable divided by its degrees of freedom. If Z is a standard normal variable and X is a chi-squared variable with n degrees of freedom, then $T = \frac{Z}{\sqrt{X/n}}$ follows the t-distribution with n degrees of freedom.

Q14: Implement a Metropolis-driven random walk to generate binomial deviates with n trials and success probability p. If the random walk is in state x,

then it should propose states $x-1$ and $x+1$ with equal probabilities. Check that the visited states have approximate mean np and approximate variance $np(1-p)$

```
using Random, Distributions, Plots

function metropolis_binomial(n, p, steps)
    current_state = rand(0:n)
    states = [current_state]

    for _ in 1:steps
        # random walk
        proposal = current_state + (rand(Bool) ? 1 : -1)

        # Ensure proposal is within valid range
        if proposal >= 0 && proposal <= n
            acceptance_ratio = binomial_pdf(n, p, proposal) / binomial_pdf(n, p, current_sta
            if rand() < min(1, acceptance_ratio)
                current_state = proposal
            end
        end

        push!(states, current_state)
    end

    return states
end

function binomial_pdf(n, p, k)
    return binomial(n, k) * p^k * (1-p)^(n-k)
end

# main function
seed_value = 123
Random.seed!(seed_value)
n = 10
p = 0.5
steps = 10000
states = metropolis_binomial(n, p, steps)

# Calculating mean and variance
mean_states = mean(states)
variance_states = var(states)

println("Mean: $mean_states, Expected Mean: $(n*p)")
println("Variance: $variance_states, Expected Variance: $(n*p*(1-p))")
```

```
histogram(states, bins=n+1, normalization=:probability, alpha=0.5, label="Metropolis States'

>>>Mean: 4.902509749025097, Expected Mean: 5.0
Variance: 2.549194700529944, Expected Variance: 2.5
```

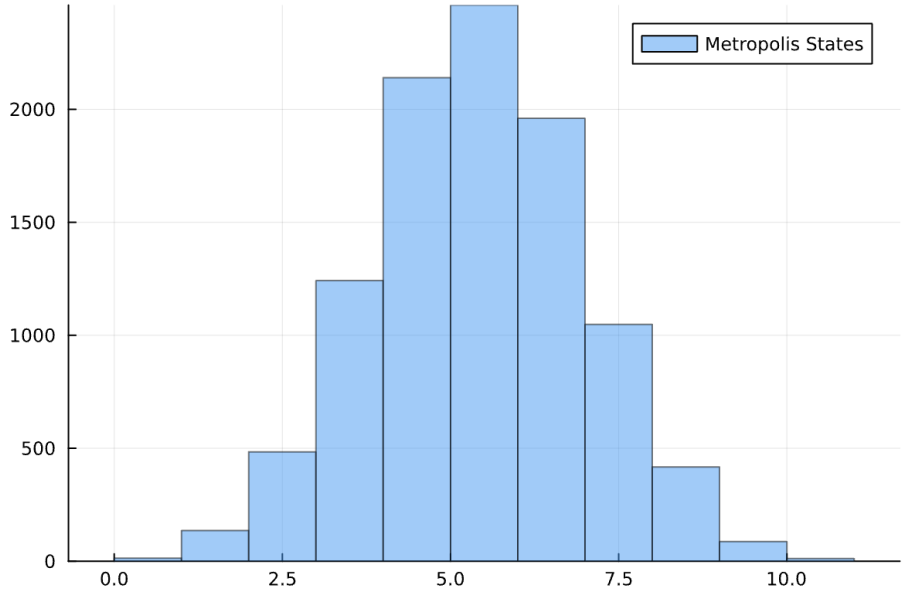

Figure 2: Binomial Variates generated from Metropolis driven random walk

Q16: Design and implement a Gibbs sampler for generating bivariate normal deviates

A Gibbs sampler is an iterative Markov Chain Monte Carlo (MCMC) algorithm that generates samples from a multivariate distribution by sampling from its conditional distributions in a cyclic fashion. For a bivariate normal distribution, the conditional distributions are univariate normal distributions.

This Julia code implements a Gibbs sampler to generate samples from a bivariate normal distribution. The gibbs_sampler function initializes two arrays, x and y, and iteratively samples from the conditional distributions of x given y and y given x. The samples are generated using the randn() function, which generates random numbers from a standard normal distribution

```
using Random, Plots
```

```
function gibbs_sampler(n::Int, mu::Vector{Float64}, sigma::Matrix{Float64})
    # Initialize variables
    x = zeros(n)
    y = zeros(n)

    # Gibbs sampler iterations
    for i in 2:n
        # Sample x conditional on y
        mu_x_given_y = mu[1] + sigma[1, 2] / sigma[2, 2] * (y[i - 1] - mu[2])
        sigma_x_given_y = sigma[1, 1] - sigma[1, 2] / sigma[2, 2] * sigma[1, 2]

        x[i] = randn() * sqrt(sigma_x_given_y) + mu_x_given_y

        # Sample y conditional on x
        mu_y_given_x = mu[2] + sigma[1, 2] / sigma[1, 1] * (x[i] - mu[1])
        sigma_y_given_x = sigma[2, 2] - sigma[1, 2] / sigma[1, 1] * sigma[1, 2]

        y[i] = randn() * sqrt(sigma_y_given_x) + mu_y_given_x
    end

    return hcat(x, y)
end

# Parameters for the bivariate normal distribution
mu = [0.0, 0.0]
sigma = [1.0 0.7; 0.7 1.0]

# Number of samples
n_samples = 1000

# Generate samples using Gibbs sampler
samples = gibbs_sampler(n_samples, mu, sigma)

# Plot the samples
plot(samples[:, 1], samples[:, 2], seriestype = :scatter, title = "Bivariate Normal Samples'
```

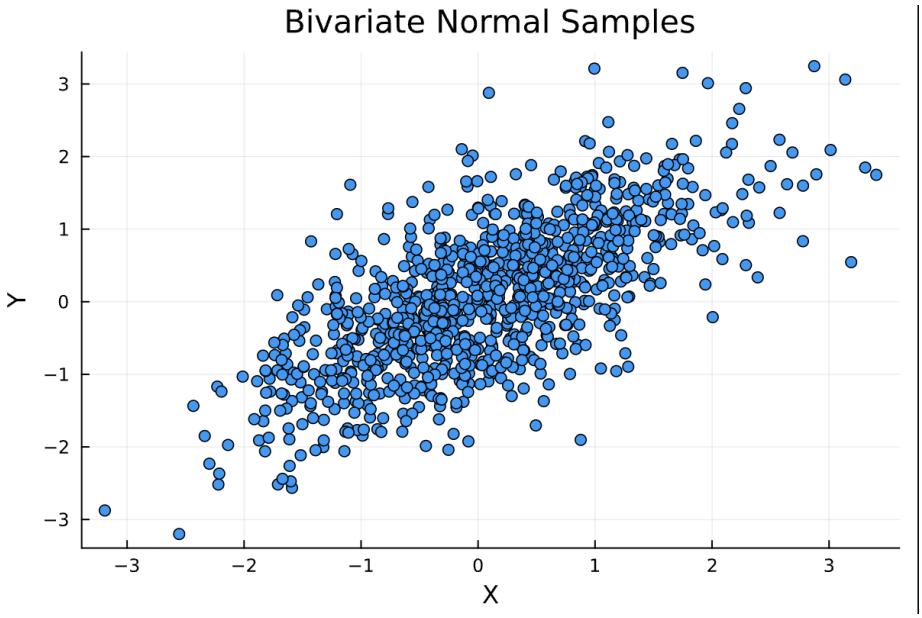

Figure 3: Bivariate Normal Samples generated from the Gibbs sampler