```
1 using Plots
```

Algoritmos Computacionales: Proyecto Intermedio

Elaborado por:

Espinosa Giron Natalia Arlette

Hernández García Viridiana

Ejercicio 1: Aproximación de π

1.1 Escribe un algoritmo que estime el valor de π y que te permita visualizar algo similar al gráfico de la Figura 1, asegurate de incluir el conteo del número de puntos rojos, número de puntos totales, y la respectiva estimación de π

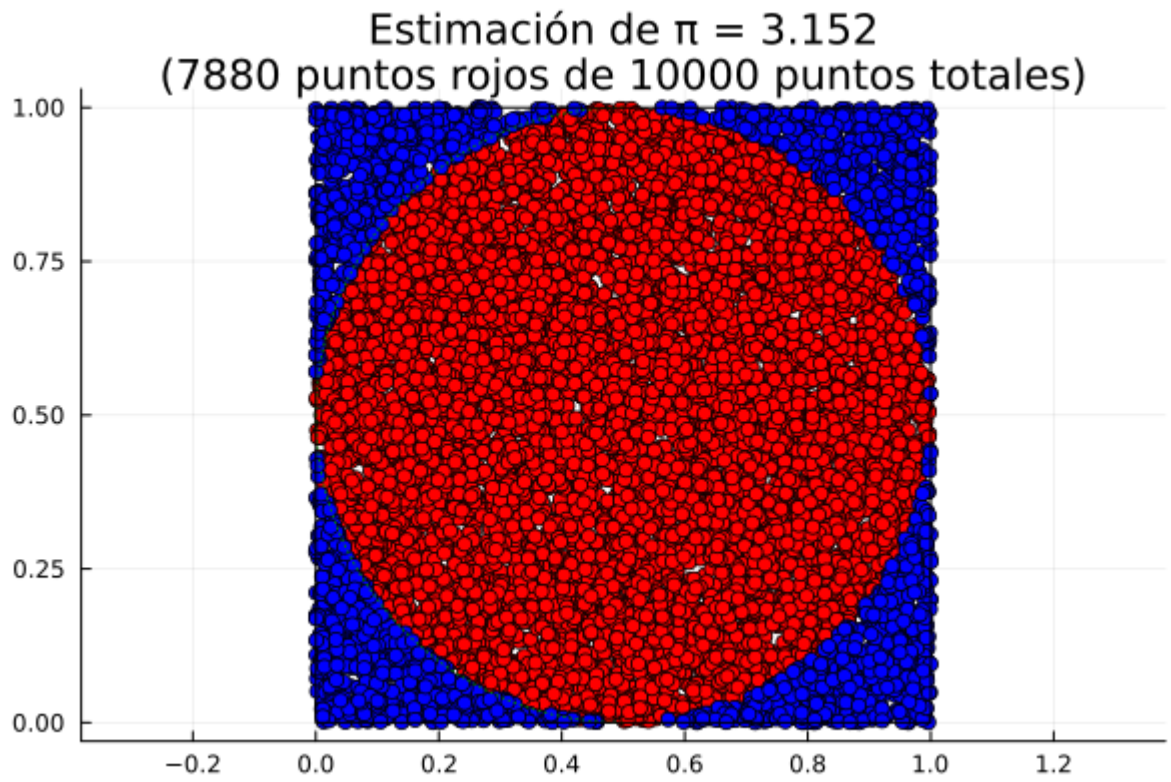
Implementamos un código donde se generan 10,000 puntos aleatorios dentro de un cuadrado de lado 1, se cuentan cuántos de ellos caen dentro de un círculo de radio 0.5. A partir de la cantidad de puntos dentro del círculo (puntos rojos) y de la cantidad de puntos totales, se estima el valor de π y se dibujan los puntos generados junto con el círculo y el cuadrado correspondientes.

```
10000
```

```
begin

# Generar 10,000 puntos aleatorios dentro del cuadrado de lado 1
n = 10000
x = rand(n)
y = rand(n)

# Contar cuántos puntos caen dentro del círculo de radio 0.5
r = 0.5
inside_circle = (x .- 0.5).^2 .+ (y .- 0.5).^2 .<= r^2
num_inside = sum(inside_circle)
num_total = length(x)

end
```

```
3.152
```

```
begin

# Calcular la estimación de π
pi_estimate = 4 * num_inside / num_total

end
```




```
begin

# Visualizar los puntos generados junto con el círculo y el cuadrado
scatter(x[inside_circle], y[inside_circle], color=:red, aspect_ratio=:equal, legend=false)
scatter!(x[.!inside_circle], y[.!inside_circle], color=:blue)
plot!(x -> sqrt(r^2 - (x - 0.5)^2) + 0.5, 0, r, color=:green, linestyle=:dot)
plot!(x -> -sqrt(r^2 - (x - 0.5)^2) + 0.5, 0, r, color=:green, linestyle=:dot)
plot!([0, 1, 1, 0, 0], [0, 0, 1, 1, 0], color=:black, aspect_ratio=:equal)
title!("Estimación de π = $(pi_estimate)\n($num_inside puntos rojos de $num_total puntos totales)")

end
```

1.2 En promedio ¿cuantos puntos necesitas generar para obtener una precision de ±0.01?

La precisión de ±0.01 se refiere a la diferencia absoluta entre la estimación de π obtenida y el valor real de π. Sabemos que el valor real de π es aproximadamente 3.14159265358979323846.

Entonces, si queremos una precisión de ±0.01, la diferencia absoluta entre la estimación y el valor real no puede ser mayor que 0.01. Esto significa que la estimación debe estar en el rango [π - 0.01, π + 0.01].

Para ello, consideramos la varianza de la estimación de π que es:

$$\sigma^2 = \pi/4 * (1 - \pi/4)/n$$

donde n es el número de puntos generados.

```
estimate_pi (generic function with 2 methods)
```

```
begin

    function estimate_pi(precision::Float64=0.01)
    #Valor real de pi
    real_pi = pi

    #Inicializar la estimación y el número de puntos generados
    pi_estimate = 0.0
    num_points = 0

    #Generar puntos aleatorios hasta que se alcanza la precisión deseada
    while abs(pi_estimate - real_pi) > precision
         #Generar un punto aleatorio
        x = rand()
        y = rand()

        #Verificar si el punto está dentro del círculo de radio 0.5
        if (x - 0.5)^2 + (y - 0.5)^2 <= 0.25
            num_points += 1
        end

        #Actualizar la estimación de pi
        pi_estimate = 4 * num_points / (num_points + (num_points == 0))

        #Mostrar el progreso cada 1000 puntos generados
        if num_points % 1000 == 0
            println("Número de puntos generados: ", num_points)
            println("Estimación de pi: ", pi_estimate)
        end
    end

    #Mostrar la estimación final de pi y el número de puntos generados
    println("Número de puntos generados: ", num_points)
    println("Estimación final de pi: ", pi_estimate)
end
end
```

La función estimate_pi toma un argumento precision opcional que especifica la precisión deseada (por defecto, se utiliza una precisión de 0.01). La función inicializa la estimación de pi y el número de puntos generados a cero, y luego genera puntos aleatorios dentro del cuadrado de lado 1 y cuenta cuántos de ellos están dentro del círculo de radio 0.5. La estimación de pi se actualiza después de cada punto generado, y el bucle continúa hasta que la diferencia absoluta entre la estimación y el valor real de π es menor que la precisión deseada.

La función también muestra el progreso cada 1000 puntos generados y finalmente muestra la estimación final de π y el número total de puntos generados. Para utilizar la función, simplemente llamamos a estimate_pi sin argumentos para utilizar la precisión por defecto de 0.01, o podemos especificar la precisión deseada como un argumento, por ejemplo:

```
#begin
    #estimate_pi(0.01)
#end
```

Cabe recalcar que ejecutar la celda anterior puede ser computacionalmente demandante

Además, implementamos el código siguiente, el cuál genera una cantidad creciente de puntos aleatorios hasta que la estimación de π alcanza una precisión de ±0.01. La función generar_puntos se encarga de generar los puntos y calcular la estimación de π correspondiente.

La precisión deseada se especifica mediante la variable error_rel_max.

MethodError: no method matching -(::LinearAlgebra.Transpose{Float64, Vector{Float64}}, ::Irrational{:π})

```
For element-wise subtraction, use broadcasting with dot syntax: array .- scalar
Closest candidates are:
-(!Matched::Base.TwicePrecision, ::Number) at twiceprecision.jl:304
-(!Matched::ColorTypes.AbstractGray{Bool}, ::Number) at
C:\Users\Asus\.julia\packages\ColorVectorSpace\QI5vM\src\ColorVectorSpace.jl:342
-(!Matched::ColorTypes.AbstractGray, ::Number) at
C:\Users\Asus\.julia\packages\ColorVectorSpace\QI5vM\src\ColorVectorSpace.jl:340
...
```

1. **top-level scope** @ *Local: 25*

```
begin
    function generar_puntos(num_puntos)
    # Generar num_puntos puntos aleatorios dentro del cuadrado de lado 1
    x = rand(num_puntos)
    y = rand(num_puntos)

    # Contar cuántos puntos caen dentro del círculo de radio 0.5
    r = 0.5
    inside_circle = (x .- 0.5).^2 .+ (y .- 0.5).^2 .<= r^2
    num_inside = sum(inside_circle)

    # Calcular la estimación de π
    pi_est = 4 * num_inside / num_puntos

    return pi_est
end

error_rel_max = 0.01
pi_est = pi - 0.01*pi

nume_puntos = ceil((pi/0.01)^2)

pi_est = generar_puntos(num_puntos)

while abs(pi_est - pi) > error_rel_max*pi
    num_puntos += 1
    pi_est = generar_puntos(num_puntos)
end
end
```

```
Assignment to `pi_est` in soft scope is ambiguous because a global variable by th
e same name exists: `pi_est` will be treated as a new local. Disambiguate by usin
g `local pi_est` to suppress this warning or `global pi_est` to assign to the exi
sting global variable.
```

```
begin
println("Se necesitan $num_puntos puntos para obtener una precisión de ±0.01 en la estimación de π.")
end
```

```
Se necesitan [10, 100, 1000, 10000, 100000, 1000000] puntos para obtener una precisión de ±0.01 en la estimación de π.
```

El código implementa el método de Montecarlo para estimar el valor de pi con una precisión relativa máxima de 0.01. Primero se define una función llamada generar_puntos que genera un número determinado de puntos aleatorios dentro de un cuadrado de lado 1 y luego cuenta cuántos de esos puntos están dentro del círculo de radio 0.5 centrado en el origen. Con esta información, se calcula una estimación de pi.

Luego, se define una precisión relativa máxima permitida del 0.01 y se inicializa una estimación de pi con un error relativo del 1%. Se calcula el número de puntos necesarios para que la precisión relativa de la estimación sea menor o igual al 0.01 definido, y se genera una estimación de pi a partir de esa cantidad de puntos. Si la diferencia absoluta entre la estimación de pi y pi verdadero es mayor que la precisión relativa máxima permitida, se incrementa el número de puntos y se genera una nueva estimación de pi. Este proceso se repite hasta que se alcanza la precisión relativa máxima permitida.

3. Realiza una grafica del error de la estimacion en funcion del numero de puntos comparando contra el valor predeterminado de π de Julia (que se obtiene llamando a la constante pi)

Para ello implementamos un código en donde se describe lo siguiente:

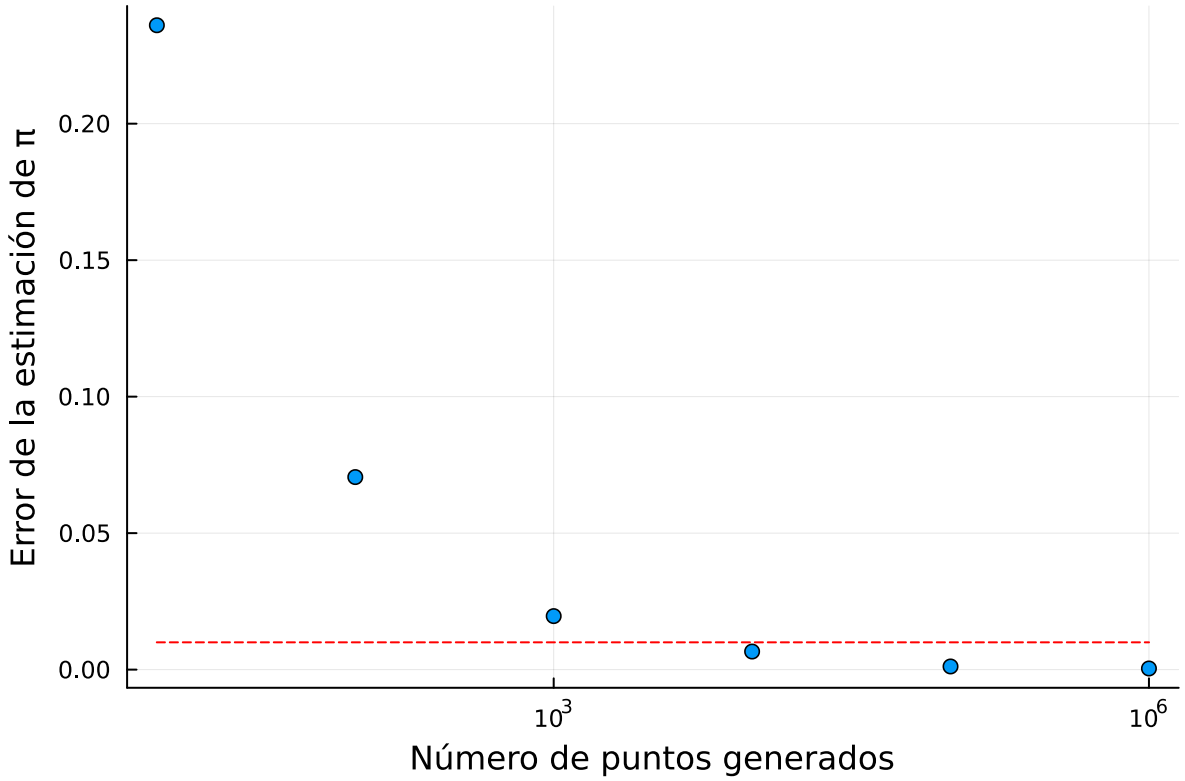
La función `error_estimacion_pi` recibe como parámetro la estimación de pi realizada en cada iteración y calcula el error absoluto de dicha estimación con respecto al valor real de pi.

El arreglo `num_puntos` contiene las cantidades de puntos aleatorios que se generarán en cada iteración para estimar pi, desde 10 hasta 10^6.

El arreglo `errores` se utiliza para almacenar los errores absolutos de las estimaciones de pi en cada iteración.

El bucle `for` itera sobre cada cantidad de puntos especificada en `num_puntos`. Dentro de este bucle, se inicializa el contador `puntos_dentro` y se generan `n` puntos aleatorios. Para cada punto, se evalúa si está dentro del círculo unitario y se incrementa el contador `puntos_dentro` si es así. Luego, se calcula la estimación de pi a partir de la cantidad de puntos dentro del círculo y se calcula el error absoluto de la estimación de pi utilizando la función `error_estimacion_pi`. Finalmente, el error se almacena en el arreglo `errores`.

```
begin
function error_estimacion_pi(pi_estimado::Float64)
    return abs(pi - pi_estimado)/pi # Calcula el error absoluto de la estimación de pi
end

num_puntos = [10^i for i in 1:6] # Crea un arreglo con la cantidad de puntos que se van a generar en cada iteración
errores = [] # Crea un arreglo para almacenar los errores en cada iteración

# Itera sobre cada cantidad de puntos
for n in num_puntos
    puntos_dentro = 0 # Contador para los puntos dentro del círculo
    # Genera n puntos aleatorios y cuenta cuántos están dentro del círculo
    for i in 1:n
        A = rand()
        B = rand()
        if A^2 + B^2 <= 1
            puntos_dentro += 1
        end
    end
    pi_estimado = 4*puntos_dentro/n # Calcula la estimación de pi a partir de la cantidad de puntos dentro del círculo
    error = error_estimacion_pi(pi_estimado) # Calcula el error absoluto de la estimación de pi
    push!(errores, error) # Almacena el error en el arreglo correspondiente
end

end
```




```
begin
    scatter(num_puntos, errores, xaxis=:log10, xlabel="Número de puntos generados",
    ylabel="Error de la estimación de π", legend=false)
plot!([num_puntos[1], num_puntos[end]], [0.01, 0.01], linestyle=:dash, color=:red,
label="Precisión objetivo ±0.01")

end
```

El código realiza una visualización de los errores de la estimación de π en función de la cantidad de puntos generados.

La función `scatter` genera un gráfico de dispersión donde el eje x es la cantidad de puntos generados (en escala logarítmica) y el eje y es el error absoluto de la estimación de π para cada cantidad de puntos.

El comando `plot!` agrega una línea horizontal en el valor de 0.01 en el eje y, que representa la precisión objetivo de la estimación (±0.01).

En resumen, el gráfico nos muestra cómo el error de la estimación de π disminuye a medida que aumenta la cantidad de puntos generados, y cómo eventualmente se alcanza la precisión objetivo de ±0.01.