Exercise: Vectorization of MC Simulation for π

```
clear; clc;

n = 1e5;
x = rand(n, 1);
y = rand(n, 1);
m = sum(x .^ 2 + y .^ 2 < 1);
result = 4 * m / n
```

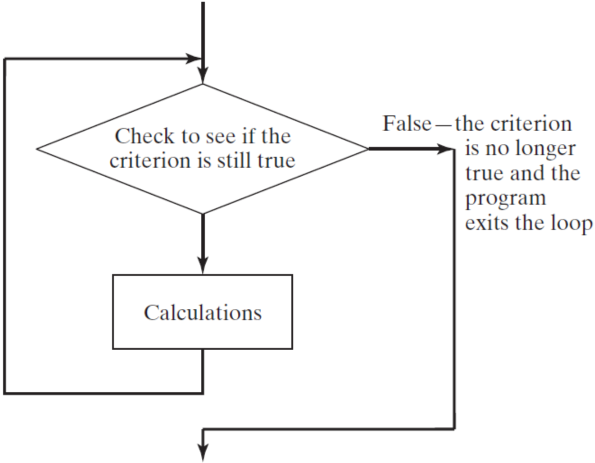
- More clear and faster!!!

while Loops

- The while loops are used to repeat the instructions until the continuation criterion is not satisfied.

```
while criterion
    % body
end
```

- Be aware that the if statement executes only once; you should use the while loop if you want to repeat some actions.


Check to see if the criterion is still true
False — the criterion is no longer true and the program exits the loop
Calculations

Example: Compounding

- Let balance be the initial amount of some investment, and r be the annualized return rate.
- Write a program which calculates the holding years when this investment doubles it value.

Solution

- In this case, we don't know how many iterations we need before the loop.

```
clear; clc;

balance = 100;
r = 0.01;
goal = 200;

holding_years = 0;
while balance < goal
    balance = balance * (1 + r);
    holding_years = holding_years + 1;
end
holding_years
```

- Note that the criterion is evaluated to continue the loop.

Infinite Loops

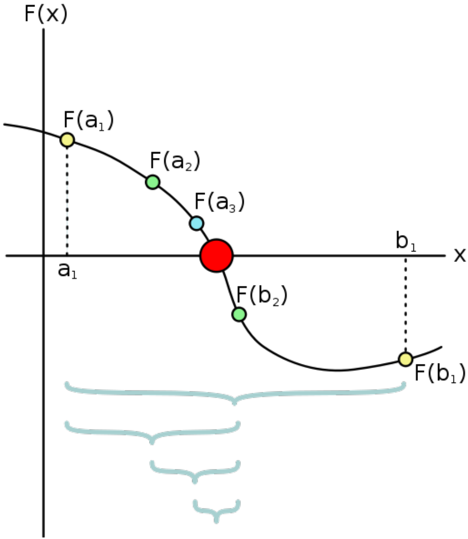
```
while true
    disp("Press ctrl+c to stop me!!!");
end
```

- Note that your program can terminate the program by pressing ctrl+c.

More Exercises (Optional)

- Let $a > b$ be two any positive integers.
- Write a program which calculates the remainder of a divided by b.
  - Do not use **mod**(a, b).
- Write a program which determines the greatest common divisor (GCD) of a and b.
  - Do not use **gcd**(a, b).

Numerical Example: Bisection Method for Root-Finding



Problem Formulation

Input

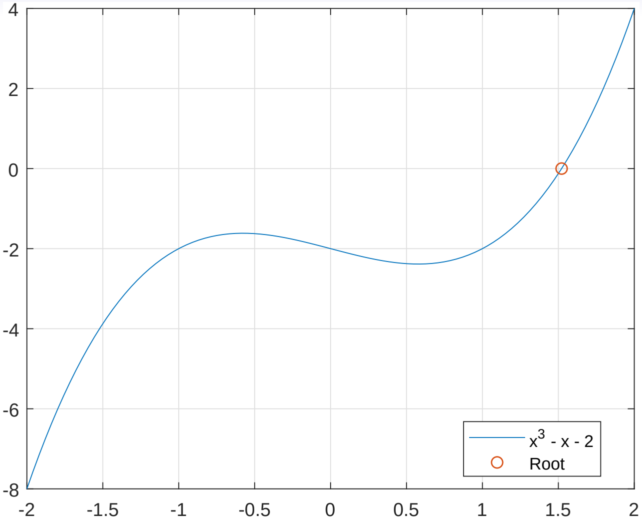
- Target function $f(x) = x^3 - x - 2$.
- Initial search interval $[a, b] = [1, 2]$.
- Error tolerance $\epsilon = 1e - 9$.

Output

- The approximate root $\hat{r}$.

Solution

```
clear; clc;

a = 1; b = 2; eps = 1e-9;

while b - a > eps

    c = (a + b) / 2;
    fa = a * a * a - a - 2;
    fc = c * c * c - c - 2;

    if fa * fc < 0
        b = c;
    else
        a = c;
    end

end
root = c
residual = fc
```



$$c = 1.52137970691547$$

"All science is dominated by the idea of approximation."

– Bertrand Russell (1872–1970)

Jump Statements

- A break statement terminates a for or while loop immediately.
  - Aka early termination.
- A continue statement skips instructions behind it and start the next iteration.
  - Directly jump to the very beginning of the loop; still in the loop.
- Notice that the break and continue statements must be conditional.

Example: Primality Test[1]

- Let x be any positive integer larger than 2 as input.
- Then x is a prime number if $\forall y \in \{2, 3, \ldots, x-1\}$, y is not a divisor of x, denoted by $y \nmid x$.
- In other words, x is called a composite number if $\exists y \in \{2, 3, \ldots, x-1\}$, $y \mid x$.
- Now write a program which determines if x is a prime number.

[1] Also see Manindra Agrawal, Neeraj Kayal, Nitin Saxena (2002).

```matlab
clear; clc;

x = input('Enter x > 2? ');
isPrime = true; % a flag, true if the number is prime
for y = 2 : sqrt(x)
    if mod(x, y) == 0
        isPrime = false;
        break;
    end
end

if isPrime
    disp([num2str(x) ' is a prime number.']);
else
    disp([num2str(x) ' is a composite number.']);
end
```

Equivalence: for and while Loops

- Whatever you can do with a for loop can be done with a while loop, and vice versa.

```
clear; clc;

balance = 100; goal = 200; r = 0.01;

for years = 1 : inf % inf: a huge but finite integer
    balance = balance * (1 + r);
    if balance >= goal
        break;
    end
end
years
```

- For another example,

```
clear; clc;

x = input("Enter x > 2? ");

isPrime = true; y = 2;
while isPrime && y < x
    isPrime = mod(x, y);
    y = y + 1;
end

if isPrime
    disp(num2str(x) + " is a prime number.");
else
    disp(num2str(x) + " is a composite number.");
end
```

Nested Loops

- Write a program which outputs the following patterns:

```
*          *****          *    *****
**         ****          **     ****
***        ***          ***      ***
****       **          ****       **
*****      *          *****        *

 (a)        (b)        (c)        (d)
```

- You may use **fprintf**("*") and **fprintf**("\n") to print a single star and break a new line, respectively.

```
clear; clc;

% case (a)
for i = 1 : 5
    for j = 1 : i
        fprintf("*");
    end
    fprintf("\n");
end
```

Exercise: $e \sim 2.7183$

- Write a program to estimate the Euler constant by Monte Carlo simulation.
- It can be done as follows.
- Let N be the number of iterations.
- For each iteration, find the minimal number n so that $\sum_{i=1}^{n} r_i > 1$ where r_i is the random variable following the standard uniform distribution (you can simply use **rand**).
- Then e is the average of n.

Special Issue: Sort

```
>> stocks = {"GOOG", 15;
             "TSMC", 12;
             "AAPL", 18};
>> [~, idx] = sort([stocks{:, 2}], "descend")

idx =

     3     1     2

>> stocks = stocks(idx, :)

stocks =

    "AAPL"    [18]
    "GOOG"    [15]
    "TSMC"    [12]
```

Programming Exercise: Sorting Algorithm[2]

- Let A be any array.
- Write a program which outputs the sorted array of A (in ascending order).
- For example, $A = [5, 4, 1, 2, 3]$.
- Then the sorted array is $[1, 2, 3, 4, 5]$.

[2] See `https://visualgo.net/sorting`.

Special Issue: Random Permutation

- Use **randperm** to generate an index array with a random order.

```
>> A = ["Matlab", "Python", "Java", "C++"];
>> idx = randperm(length(A))

idx =

     3     1     2     4

>> A(idx)

ans =

  1x4 string array

    "Java"    "Matlab"    "Python"    "C++"
```

"Exploring the unknown requires tolerating uncertainty."

– Brian Greene

"I can live with doubt, and uncertainty, and not knowing. I think it is much more interesting to live not knowing than have answers which might be wrong."

– Richard Feynman

Speedup: Vectorization (Revisited)[3]

- Vector in, vector out.

```
>> x = randi(100, 1, 5)

x =

    88    30    90    73    82

>> dx = diff(x)

dx =

   -58    60   -17     9
```

[3] More about vectorization.

Advantages from Vectorization

- Appearance: vectorized mathematical code appears more like the mathematical expressions found in textbooks, making the code easier to understand.
- Less error prone: without loops, vectorized code is often shorter.
  - Fewer lines of code mean fewer opportunities to introduce programming errors.
- Performance: vectorized code often runs much faster than the corresponding code containing loops.

Performance Analysis: Profiling

- Use a timer to measure your performance.[4]
  - In newer version, press the button *Run and Time*.
- Identify which functions are consuming the most time.
- Know why you are calling them and then look for alternatives to improve the overall performance.

[4] Note that the results may differ depending on the difference of run-time environments, so make sure that you benchmark the algorithms on the same conditions.

tic & toc

- The command **tic** makes a stopwatch timer start.
- The command **toc** returns the elapsed time from the stopwatch timer started by **tic**.

```
>> tic
>> toc
Elapsed time is 0.786635 seconds.
>> toc
Elapsed time is 1.609685 seconds.
>> toc
Elapsed time is 2.417677 seconds.
```

Selected Performance Suggestions[5]

- Preallocate arrays.
  - Instead of continuously resizing arrays, consider preallocating the maximum amount of space required for an array.
- Vectorize your code.
- Create new variables if data type changes.
- Use functions instead of scripts.
- Avoid overloading Matlab built-in functions.

[5] See Techniques for Improving Performance.

Programming Exercise: A Benchmark

- Let $N = 1e1, 1e2, 1e3, 1e4, 1e5$.
- Write a program which produces a benchmark for the following three cases:
  - Generate an array of $1 : N$ by dynamically resizing the array.
  - Generate an array of $1 : N$ by allocating an array of size N and filling up sequentially.
  - Generate an array of $1 : N$ by vectorization.

Analysis of Algorithms (Optional)

- For one problem, there exist various algorithms (solutions).
- We then compare these algorithms for various considerations and choose the most appropriate one.
- In general, we want efficient algorithms.
- Except for real-time performance analysis, could we predict before the program is completed?
- Definitely yes.

Growth Rate

- Now we use $f(n)$ to denote the growth rate of time cost as a function of n.
  - In general, n refers to the data size.
- For simplicity, assume that every instruction (e.g. $+ - \times \div$) takes 1 unit of computation time.
- Find $f(n)$ for the following problem.
  - Sum(n): ?
  - Triangle(n): ?

O-notation[6]

- In math, O-notation describes the limiting behavior of a function, usually in terms of simple functions.
- We say that

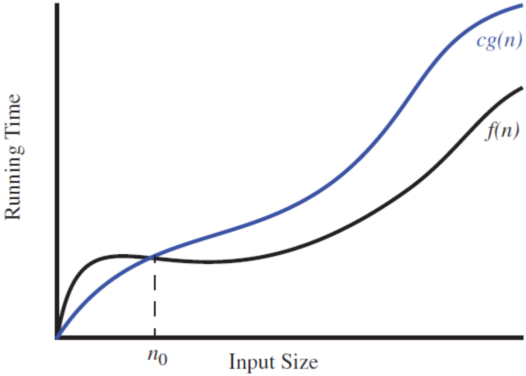
$$f(n) \in O(g(n)) \text{ as } n \to \infty$$

if and only if $\exists c > 0, n_0 > 0$ such that

$$|f(n)| \leq c|g(n)| \quad \forall n \geq n_0.$$

- So $O(g(n))$ is a collection featured by a simple function $g(n)$.
- We use $f(n) \in O(g(n))$ to denote that $f(n)$ is one instance of $O(g(n))$.

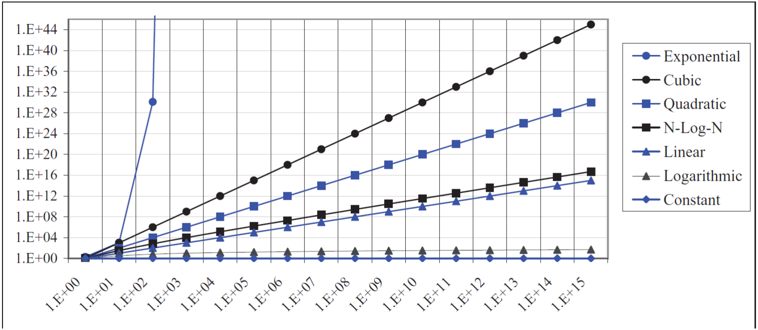
[6] See https://en.wikipedia.org/wiki/Big_O_notation.



- Big-O is used for the asymptotic upper bound of time complexity of algorithm.
- In layman's term, Big-O describes the worst case of this algorithm.

- For example, $8n^2 - 3n + 4 \in O(n^2)$.
  - For large n, you could ignore the last two terms. (Why?)
  - It is easy to find a constant $c > 0$ so that $cn^2 > 8n^2$, say $c = 9$.
  - Hence the statement is proved.
- Also, $8n^2 - 3n + 4 \in O(n^3)$ but we seldom say this. (Why?)
- However, $8n^2 - 3n + 4 \notin O(n)$. (Why?)
- What is this analysis related to the algorithm?
- Any insight?

Common Simple Functions[7]



constant	*logarithm*	*linear*	*n-log-n*	*quadratic*	*cubic*	*exponential*
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

[7] See Table 4.1 and Figure 4.2 in Goodrich and etc, p. 161.

Remarks

- We often make a trade-off between time and space.
  - Unlike time, we can reuse memory.
  - Users are sensitive to time.
- Playing game well is hard.[8]
- Solve the problem P ?= NP, which is one of Millennium Prize Problems.[9]

[8] See https://en.wikipedia.org/wiki/Game_complexity.
[9] See https://en.wikipedia.org/wiki/P_versus_NP_problem.

"All roads lead to Rome."

– Anonymous

"但如你根本並無招式，敵人如何來破你的招式？"

– 風清揚。笑傲江湖。第十回。傳劍