Math 124 - Programming for Mathematical Applications

UC Berkeley, Spring 2023

Homework 3

Due Wednesday, February 8

Problem 1

Define the matrix `A = [1:4  5:8  ones(Int64,4)]` .

Predict the result of performing the following operations in Julia (before checking your answers by running them). Note that the lines are meant to be run one-by-one in the same workspace, so when you e.g. change an array this will affect the subsequent statements.

a. `x = A[3,:]`

b. `B = A[2:2,1:3]`

c. `A[1,1] = 9 + A[2,3]`

d. `A[1:3,2:3] = [0 0; 0 0; 0 0]`

e. `A[1:2,2:3] = [1 3; 1 3]`

f. `y = A[1:4, 3]`

g. `A = [A [2 1 7; 7 4 5; ones(Int64,2,3)]]`

h. `C = A[[1,3],2]`

i. `D = A[[1,2,4],[1,3,4]]`

In [1]:
```
#a. returns x as the 3rd row vector of A:
A = [1:4  5:8  ones(Int64,4)]
x = A[3,:]
```

Out[1]:
```
3-element Vector{Int64}:
 3
 7
 1
```

In [2]:
```
#b. Returns another matrix B which is the second row vector of A:
A = [1:4  5:8  ones(Int64,4)]
B = A[2:2,1:3]
```

Out[2]:
```
1×3 Matrix{Int64}:
 2  6  1
```

In [3]:
```
#c. Changes the entry in the first row, first column of A to 9 + the e
A = [1:4  5:8  ones(Int64,4)]
A[1,1] = 9 + A[2,3]
A
```

Out[3]:
```
4×3 Matrix{Int64}:
 10  5  1
  2  6  1
  3  7  1
  4  8  1
```

In [4]:
```
#d. Replaces the top three entries in the last two columns of A with z
A = [1:4  5:8  ones(Int64,4)]
A[1:3,2:3] = [0 0; 0 0; 0 0]
A
```

Out[4]:
```
4×3 Matrix{Int64}:
 1  0  0
 2  0  0
 3  0  0
 4  8  1
```

In [5]:
```
#e. Replaces the top two entries in the second column by 1's and the t
A = [1:4  5:8  ones(Int64,4)]
A[1:2,2:3] = [1 3; 1 3]
A
```

Out[5]:
```
4×3 Matrix{Int64}:
 1  1  3
 2  1  3
 3  7  1
 4  8  1
```

In [6]:
```
#f. Assigns the third column vector of A to y:
A = [1:4  5:8  ones(Int64,4)]
y = A[1:4, 3]
```

Out[6]:
```
4-element Vector{Int64}:
 1
 1
 1
 1
```

In [7]:
```
#g. horizontal concatenation of a matrix with A with 2, 1, 7 in top ro
A = [1:4  5:8  ones(Int64,4)]
A = [A [2 1 7; 7 4 5; ones(Int64,2,3)]]
A
```

Out[7]:
```
4×6 Matrix{Int64}:
 1  5  1  2  1  7
 2  6  1  7  4  5
 3  7  1  1  1  1
 4  8  1  1  1  1
```

In [8]:
```
#h. Sets C to have the first and third rows of the second column of A:
A = [1:4  5:8  ones(Int64,4)]
C = A[[1,3],2]
C
```

Out[8]:
```
2-element Vector{Int64}:
 5
 7
```

In [9]:
```
#i. Sets D to have the first, second, fourth row of A of the first, th
A = [1:4  5:8  ones(Int64,4)]
D = A[[1,2,4],[1,3,4]]
D
#Nvm doesn't work because the matrix does not have 4 rows and columns.
```

```
BoundsError: attempt to access 4×3 Matrix{Int64} at index [[1, 2, 4],
[1, 3, 4]]

Stacktrace:
 [1] throw_boundserror(A::Matrix{Int64}, I::Tuple{Vector{Int64}, Vect
or{Int64}})
   @ Base ./abstractarray.jl:651
 [2] checkbounds
   @ ./abstractarray.jl:616 [inlined]
 [3] _getindex
   @ ./multidimensional.jl:831 [inlined]
 [4] getindex(::Matrix{Int64}, ::Vector{Int64}, ::Vector{Int64})
   @ Base ./abstractarray.jl:1170
 [5] top-level scope
   @ In[9]:3
 [6] eval
   @ ./boot.jl:360 [inlined]
 [7] include_string(mapexpr::typeof(REPL.softscope), mod::Module, cod
e::String, filename::String)
   @ Base ./loading.jl:1116
```

Problem 2(a)

- Create a vector x of the numbers $1, 0.99, 0.98, 0.97, \ldots 0.01, 0$ in this order
- Using x, create a vector y defined by the elementwise function $y_i = \sin 2\pi x_i$ for each element in x

In [3]:
```
x = 1:-0.01:0
y = sin.(2π*x)
```

Out[3]:
```
101-element Vector{Float64}:
 -2.4492935982947064e-16
 -0.06279051952931326
 -0.12533323356430465
 -0.18738131458572468
 -0.24868988716485535
 -0.3090169943749476
 -0.36812455268467786
 -0.425779291565073
 -0.4817536741017153
 -0.5358267949789971
 -0.5877852522924734
 -0.6374239897486896
 -0.684547105928689
   ⋮
  0.6374239897486896
  0.5877852522924731
  0.5358267949789967
  0.4817536741017153
  0.42577929156507266
  0.3681245526846779
  0.3090169943749474
  0.2486898871648548
  0.1873813145857246
  0.12533323356430426
  0.06279051952931337
  0.0
```

Problem 2(b)

Given the vector y, create:

- A vector v containing the first 25 elements of y
- A vector w containing elements of y with indices from 50 to 75
- A vector z containing elements of y with even indexes

In [4]:
```
v = y[1:25]
w = y[50:75]
z = y[2:2:length(y)]
```

Out[4]:
```
50-element Vector{Float64}:
 -0.06279051952931326
 -0.18738131458572468
 -0.3090169943749476
 -0.425779291565073
 -0.5358267949789971
 -0.6374239897486896
 -0.7289686274214116
 -0.8090169943749476
 -0.8763066800438638
 -0.9297764858882516
 -0.9685831611286311
 -0.9921147013144779
 -1.0
  ⋮
  0.9921147013144779
  0.9685831611286311
  0.9297764858882513
  0.8763066800438636
  0.8090169943749475
  0.7289686274214116
  0.6374239897486896
  0.5358267949789967
  0.42577929156507266
  0.3090169943749474
  0.1873813145857246
  0.06279051952931337
```

Problem 2(c)

Given the vector y, create:

- A vector v containing the same elements in the reverse order
- A vector w containing elements of y which are smaller than -0.2
- A vector z of the *indices* in y of elements greater than 0.5 (that is, the numbers i for which $y_i > 0.5$)

In [5]:
```
v = y[length(y):-1:1]
```

Out[5]:
```
101-element Vector{Float64}:
  0.0
  0.06279051952931337
  0.12533323356430426
  0.1873813145857246
  0.2486898871648548
  0.3090169943749474
  0.3681245526846779
  0.42577929156507266
  0.4817536741017153
  0.5358267949789967
  0.5877852522924731
  0.6374239897486896
  0.6845471059286886
  ⋮
 -0.6374239897486896
 -0.5877852522924734
 -0.5358267949789971
 -0.4817536741017153
 -0.425779291565073
 -0.36812455268467786
 -0.3090169943749476
 -0.24868988716485535
 -0.18738131458572468
 -0.12533323356430465
 -0.06279051952931326
 -2.4492935982947064e-16
```

In [8]:
```
w = []
for i = 1:length(y)
    if y[i] < -0.2
        push!(w, y[i])
    else
    end
end
w
```

Out[8]:
```
43-element Vector{Any}:
 -0.24868988716485535
 -0.3090169943749476
 -0.36812455268467786
 -0.425779291565073
 -0.4817536741017153
 -0.5358267949789971
 -0.5877852522924734
 -0.6374239897486896
 -0.684547105928689
 -0.7289686274214116
 -0.7705132427757896
 -0.8090169943749476
 -0.844327925502015
  ⋮
 -0.8090169943749473
 -0.7705132427757894
 -0.7289686274214113
 -0.6845471059286887
 -0.6374239897486896
 -0.587785252292473
 -0.5358267949789964
 -0.481753674101715
 -0.42577929156507266
 -0.3681245526846779
 -0.3090169943749473
 -0.24868988716485457
```

In [19]:
```
z = Int64[]
for i = 1:length(y)
    if y[i] > .5
        push!(z, i)
    else
    end
end
z
```

Out[19]:
```
33-element Vector{Int64}:
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
  ⋮
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
```

Problem 3

Given a matrix A, e.g. `A = [0 2 1; 3 1 0; 4 6 4; 2 0 2]` :

Problem 3(a)

- Write code to create a matrix B with 1's at locations where A has zeros and 0's elsewhere

In [9]:
```
A = [0 2 1; 3 1 0; 4 6 4; 2 0 2]
B = [0 2 1; 3 1 0; 4 6 4; 2 0 2]
for i = 1:size(A,1)
    for j = 1:size(A,2)
        if A[i,j] == 0
            B[i,j] = 1
        else
            B[i,j] = 0
        end
    end
end
B
```

Out[9]:
```
4×3 Matrix{Int64}:
 1  0  0
 0  0  1
 0  0  0
 0  1  0
```

Problem 3(b)

- Write code to create a matrix C containing all 0's except the maximum elements in each row of a (i.e. using the example matrix A, C would be `[0 2 0; 3 0 0; 0 6 0; 2 0 2]`)

In [14]:
```
C = [0 2 1; 3 1 0; 4 6 4; 2 0 2]
for i = 1:size(C,1)
    max = C[i,1]
    for j = 1:size(C,2)-1
        if max < C[i,j+1]
            max = C[i,j+1]
            C[i,j] = 0
        elseif max == C[i,j+1]
            break
        else
            C[i,j+1] = 0
        end
    end
end
C
```

Out[14]:
```
4×3 Matrix{Int64}:
 0  2  0
 3  0  0
 0  6  0
 2  0  2
```

Problem 4

Write a function `tridiag(a,b,c)` to create the following *tridiagonal* matrix:

$$A = \begin{bmatrix} b_1 & c_1 & & & 0 \\ a_1 & b_2 & c_2 & & \\ & a_2 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_{n-1} & b_n \end{bmatrix}$$

for vectors `a` and `c` of length $n-1$ and vector `b` of length n.

Test your function using e.g. `tridiag(1:4, 11:15, 21:24)`.

In [15]:
```
function tridiag(a,b,c)
    A = zeros(Int64,length(b), length(b))
    for i = 1:length(b)
        A[i,i] = b[i]
    end
    for j = 1:length(a)
        A[j+1,j] = a[j]
    end
    for k = 1:length(c)
        A[k,k+1] = c[k]
    end
    A
end
```

Out[15]:
```
tridiag (generic function with 1 method)
```

In [16]:
```
tridiag(1:4, 11:15, 21:24)
```

Out[16]:
```
5×5 Matrix{Int64}:
 11  21   0   0   0
  1  12  22   0   0
  0   2  13  23   0
  0   0   3  14  24
  0   0   0   4  15
```

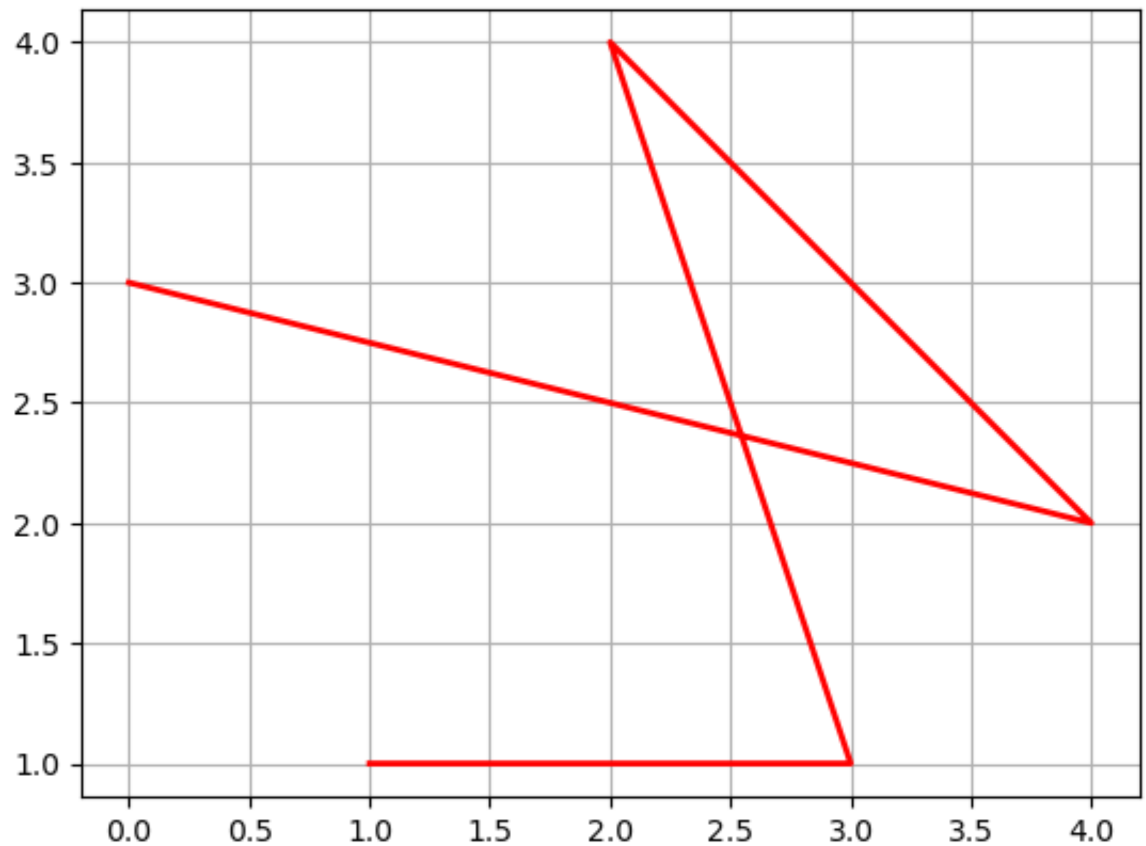
Problem 5

Make a plot connecting the coordinates: $(1, 1)$, $(3, 1)$, $(2, 4)$, $(4, 2)$ and $(0, 3)$ by a red line of linewidth 2. Use equal axis coordinates, and add grid lines.

In [11]:
```
using PyPlot
```

In [12]:
```
x_coord = [1, 3, 2, 4, 0]
y_coord = [1, 1, 4, 2, 3]
plot(x_coord, y_coord, linewidth = 2, color = "r")

grid(true)
axis("equal")
```



Out[12]:
```
(-0.2, 4.2, 0.85, 4.15)
```

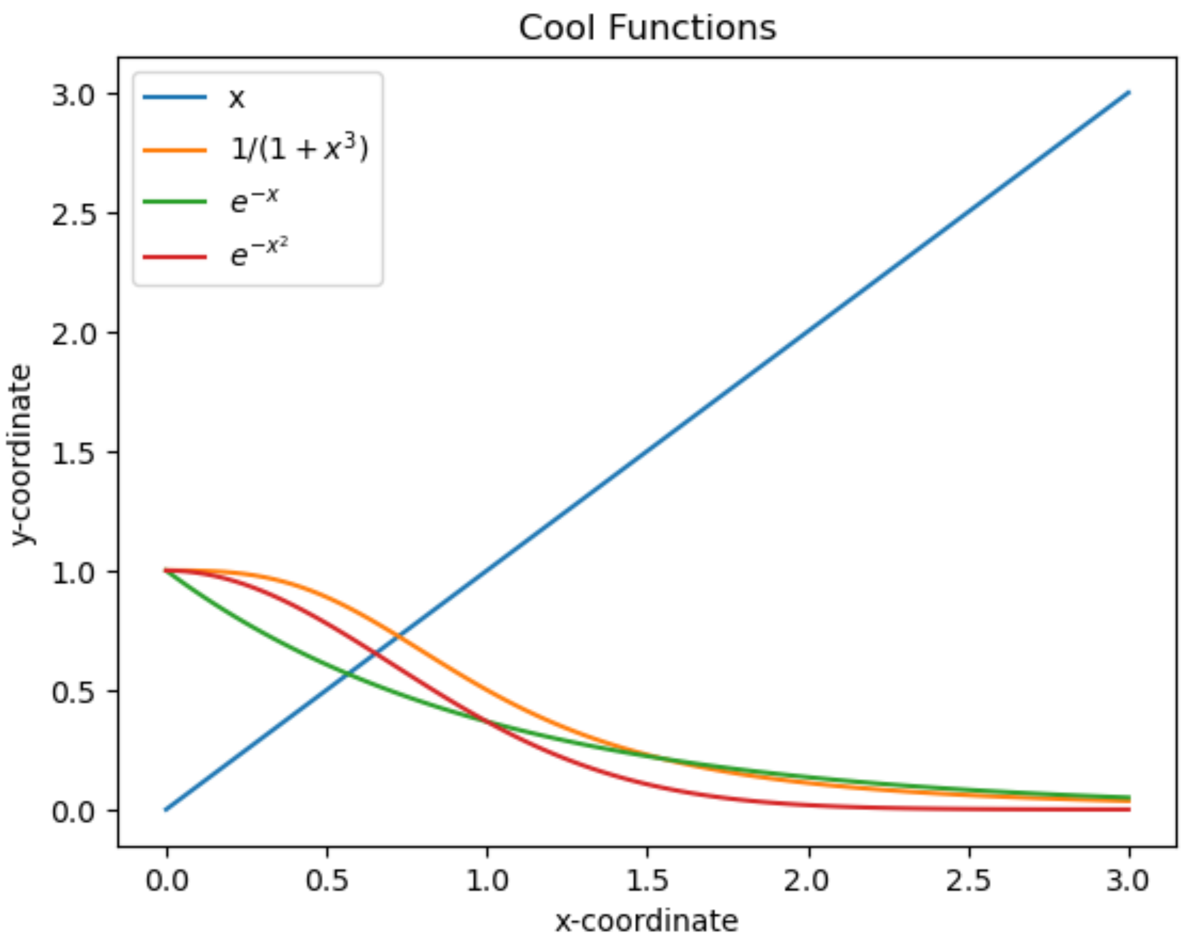
Problem 6

Plot the functions $f(x) = x$, $g(x) = 1/(1 + x^3)$, $h(x) = e^{-x}$ and $z(x) = e^{-x^2}$ over the interval $x \in [0, 3]$. Describe your plots by using the functions `xlabel` , `ylabel` , `title` and `legend` .

In [27]:
```
f(x) = x
g(x) = 1/(1 + x^3)
h(x) = exp(-x)
b(x) = exp(-x^2)
x = range(0, stop = 3, length = 100)

plot(x, f.(x))
plot(x, g.(x))
plot(x, h.(x))
plot(x, b.(x))

xlabel("x-coordinate")
ylabel("y-coordinate")
title("Cool Functions")
legend(("x", "\$ 1/(1 + x^3) \$", "\$ e^{-x} \$", "\$ e^{-x^2} \$"))
```



Out[27]:
```
PyObject <matplotlib.legend.Legend object at 0x7f506844e4c0>
```

Problem 7

(Project Euler, Problem 25)

> The Fibonacci sequence is defined by the recurrence relation:
>
> Fn = Fn−1 + Fn−2, where F1 = 1 and F2 = 1.
>
> Hence the first 12 terms will be:
>
> F1 = 1 F2 = 1 F3 = 2 F4 = 3 F5 = 5 F6 = 8 F7 = 13 F8 = 21 F9 = 34 F10 = 55 F11 = 89 F12 = 144
>
> The 12th term, F12, is the first term to contain three digits.
>
> What is the index of the first term in the Fibonacci sequence to contain 1000 digits?

To solve this problem, we need to compute with integers much larger than the built-in type `Int128`. While Julia has support for larger integers, here you will write your own implementation to solve this problem using arrays and basic arithmetics.

We will represent the large integers by an array of length n with integers between $0 \ldots 9$. This is certainly not the most efficient way to solve this problem, but it works. More precisely, the value of a large integer represented by an array X is

$$\sum_{k=1}^{n} X[k] \cdot 10^{k-1}$$

For example, with $n = 5$ the array $X = [7, 2, 4, 3, 0]$ represents the integer $x = 3427$.

We can define a helper function to print integers represented like this:

In [14]:
```
function print_bigint(X)
    for i = length(X):-1:1
        print(X[i])
    end
    println
end
```

Out[14]:
```
print_bigint (generic function with 1 method)
```

Problem 7(a)

Write a function `int_to_bigint(x,n)` which converts a regular integer `x` (e.g. of type `Int64`) to an array of length `n` as described above. For example,

```
X = int_to_bigint(34278273,10);
println(X)
print_bigint(X)
```

should output

```
[3, 7, 2, 8, 7, 2, 4, 3, 0, 0]
0034278273
```

In [15]:

```
function int_to_bigint(x,n)
    A = zeros(Int64,1,n)
    A[1,1] = x % 10
    for i = 2:n
        A[1,i] = (x ÷ 10) % 10
        x = x ÷ 10
    end
    A
end
X = int_to_bigint(34278273,10)
```

Out[15]:

```
1×10 Matrix{Int64}:
 3  7  2  8  7  2  4  3  0  0
```

Problem 7(b)

To solve the original problem, we also need a function to add two large integers represented by arrays. Implement a function `add_bigints(X,Y)` which computes the sum of the two numbers, and returns it in a new array. Use the standard addition with carry algorithm, and if there is a carry beyond the nth digit your code should run `error("Overflow")` .

For example, the code

```
x = 637465
y = 99827391
z = x + y
X = int_to_bigint(x,10)
Y = int_to_bigint(y,10)
Z = add_bigints(X,Y)
print_bigint(Z)
println(z)
```

should produce the output

```
0100464856
100464856
```

but if you change x to `9999999999` it should stop with an error.

In [16]:
```
function add_bigints(A,B)
    Z = zeros(Int64, 1, length(A))
    for i = 1:length(A)-1
        if A[i] + B[i] + Z[i] ≥ 10
            Z[i+1] += (A[i] + B[i] + Z[i]) ÷ 10
            Z[i] += (A[i] + B[i]) - Z[i+1]*10
        else
            Z[i] += A[i] + B[i]
        end
    end
    Z
end
```

Out[16]:
```
add_bigints (generic function with 1 method)
```

In [17]:
```
x = 637465
y = 99827391
z = x + y
X = int_to_bigint(x,10)
Y = int_to_bigint(y,10)
Z = add_bigints(X,Y)
print_bigint(Z)
println(z)
```

```
0100464856
100464856
```

Problem 7(c)

Finally we can solve the original Fibonacci problem. Write a function `big_fibonacci(n)` which finds the first term in the sequence to contain `n` digits, prints that number (using `print_bigint`) and returns the index.

Try your function on the original problem, that is, run `big_fibonacci(1000)` .

In [30]:
```
function big_fibonacci(n)
    X_n1 = int_to_bigint(1,n)
    X_n2 = int_to_bigint(1,n)
    Save = int_to_bigint(1,n)
    Index = 2
    while X_n2[n] == 0
        Save = X_n2
        X_n2 = add_bigints(X_n1, X_n2)
        X_n1 = Save
        Index += 1
    end
    println("x_", Index, " = ")
    print_bigint(X_n2)
end
```

Out[30]:
```
big_fibonacci (generic function with 1 method)
```

In [31]:
```
big_fibonacci(1000)
```

```
x_4782 =
107006626638275893676498058445739688508368389663215166501323520337531
452060469404062188914758248979265780469488817759195748433646667256995
951299603046126274809248218614406943305123477444275027378175308757939
166619214925918675955396642283714894311307469950343954700198543260972
306729019287052644724372611771582182554849112052501320147861296593138
179223555965745203950613755146783754322911960212993404826070617539770
684706820289548690266618543512452190036948064135744747091170761976694
569107009802439343961747410373691250323136553216477369702316775505159
517351846057995491941096777837322966579658164651390348815425631018422
419025984608800011018625555024549393711365165703944762958471454852342
595042858242530608354443542821261100899286379504800689433030977321783
486454311320576565986845628861680871869383529735064398629764066000072
356291790520705116407761481249188583094594056668833910935094445657635
766615161931775379289166158132715961687748798382182049252034847387438
47367719345127870292186362506278 16
```