MATLAB, Lab 10– Individual work

Matlab good programming habits

There are some good programming habits allowing to save computational time. First of all, the memory should be preallocated, which means that before running the loop matrix of the adequate size should be created. This can be accomplished with the use of function *zeros.*

Without memory preallocation	With memory preallocation

`n = 10000;`
`for k = 1 : n`
`x (k) = k;`
`end`
	`n = 10000;`
	`x = zeros (1, n);`
	`% mem.preallocation`
	`for k = 1 : n`
	`  x (k) = k;`
	`end`

In the first example Matlab needs to enlarge matrix size on every loop iteration. It requires moving of large amount of data which is time-consuming. Therefore it is recommended to save sufficient amount of memory by creating matrix filled with zeros.

Even with memory preallocation, loops are not the most optimal solution and should be avoided if it is only possible. Matlab is optimized to work with large sets of data stored in matrices. All matrix operators are programmed in a most efficient way, not reachable with the use of typical loops.

Task

1. Optimize the code by omitting an unnecessary loop
   a. Subtract 7 from all elements on the matrix diagonal

```
A=rand(100,100)*10;
for i=1:100
    A(i,i)=A(i,i)-7;
end
```

Code:
`A=rand(100,100)*10;` `A(1:1+size(A,1):end) = A(1:1+size(A,1):end)-7;`

   b. Count the number of elements of matrix A which are bigger than the adequate elements of matrix B

```
A=rand(100,100)*10;
B=rand(100,100)*10;
a_bigger=0;
for i=1:100
   for j=1:100
      if A(i,j)>B(i,j)
         a_bigger=a_bigger+1;
      end
   end
end
```

Code:

```
n = rand(100,100)*10;
b = rand(100,100)*10;
t = sum(n>b);
sum(t)
```

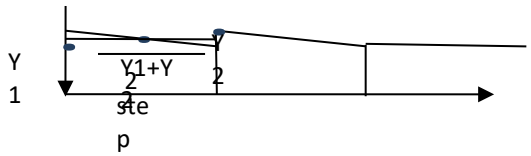
c. Create vector with sums of 100 natural numbers (so-called cumulative sum): B=[1, 1+2, 1+2+3... 1+2+3+...+n]

```
A=1:100;
B=zeros(1,100);
for i=1:100
   for j=1:i
      B(i)=B(i)+A(j);
   end
end
```

Code:

```
A=1:100;
B = cumsum(A);
disp(B);
```

d. Approximate the area under the curve y=x^2 within limits [-1,1] on 100 samples




```
X=linspace(-1,1,100);
Y=X.^2;
step=(1+1)/99;
area_val=0;
for i=1:99
   area_val=area_val+((Y(i+1)+Y(i))/2)*step;
end
```

Code:

```
x= linspace(-1,1,100);
y=x.^2;
y1= y(1:99);
```

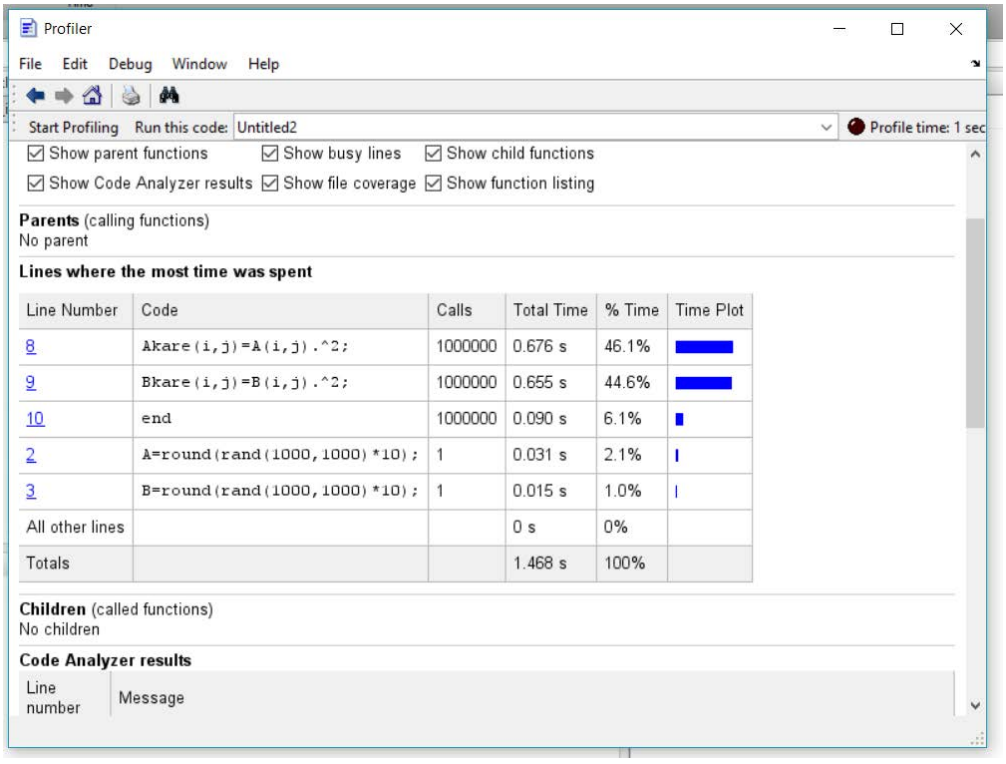
```
y2= y(2:100);
Integral=sum(0.5.*(y2+y1).*diff(x));
```

2. Create two matrices *n* x *n* with random numbers (choose n according to the speed of your computer) Then prepare algorithm adding its elements one by one in loop. Determine working time of this algorithm (could be done by profiler) in two cases: with memory preallocation and without it. Afterwards perform classic matrix operation of addition. Write similar problems for subtraction, multiplication, division and second power (all operators elementwise- .*, ./, .^). Create the following table and fill it with time required by written algorithms. Attach screenshotsof the Profiler showing your results obtained for choosen operation calculated with the use of three methods.

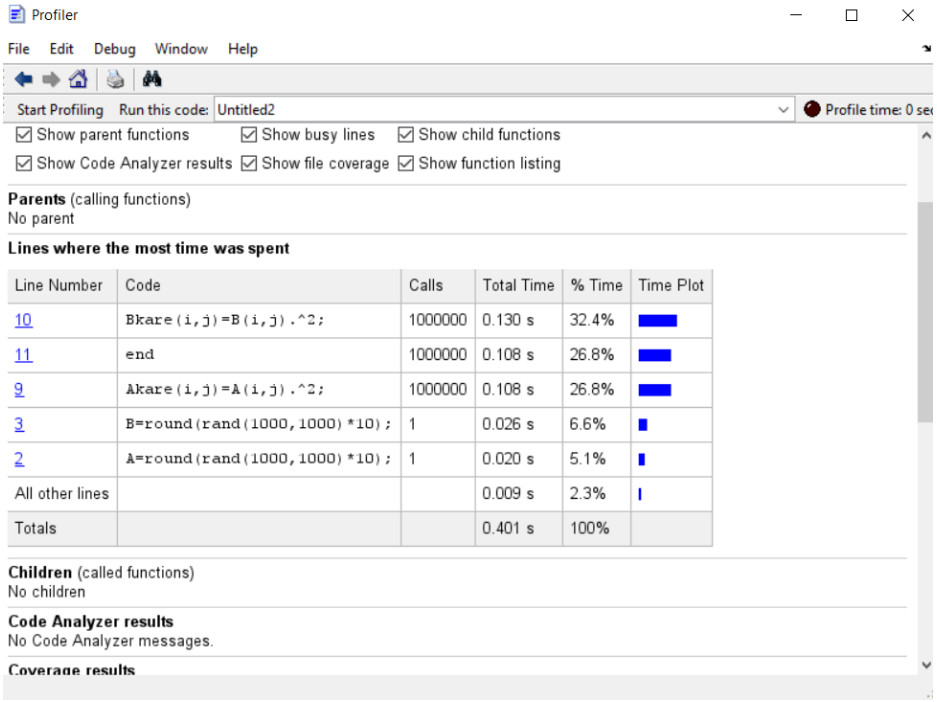
	Loop	Loop with memory preallocation	Matrix operation
Addition	0.751 s	0.258 s	0.050 s
Subtraction	0.758 s	0.260 s	0.053 s
Multiplication (.*)	0.733 s	0.264 s	0.039 s
Division (./)	0.743 s	0.269 s	0.059 s
Second power (.^)	1.468 s	0.401 s	0.065 s

My chosen operation is second power operation for each implementation.

Screenshot (loop):



Screenshot (loop+preallocation):



Screenshot (matrixoperation):