1. Forward Algorithm

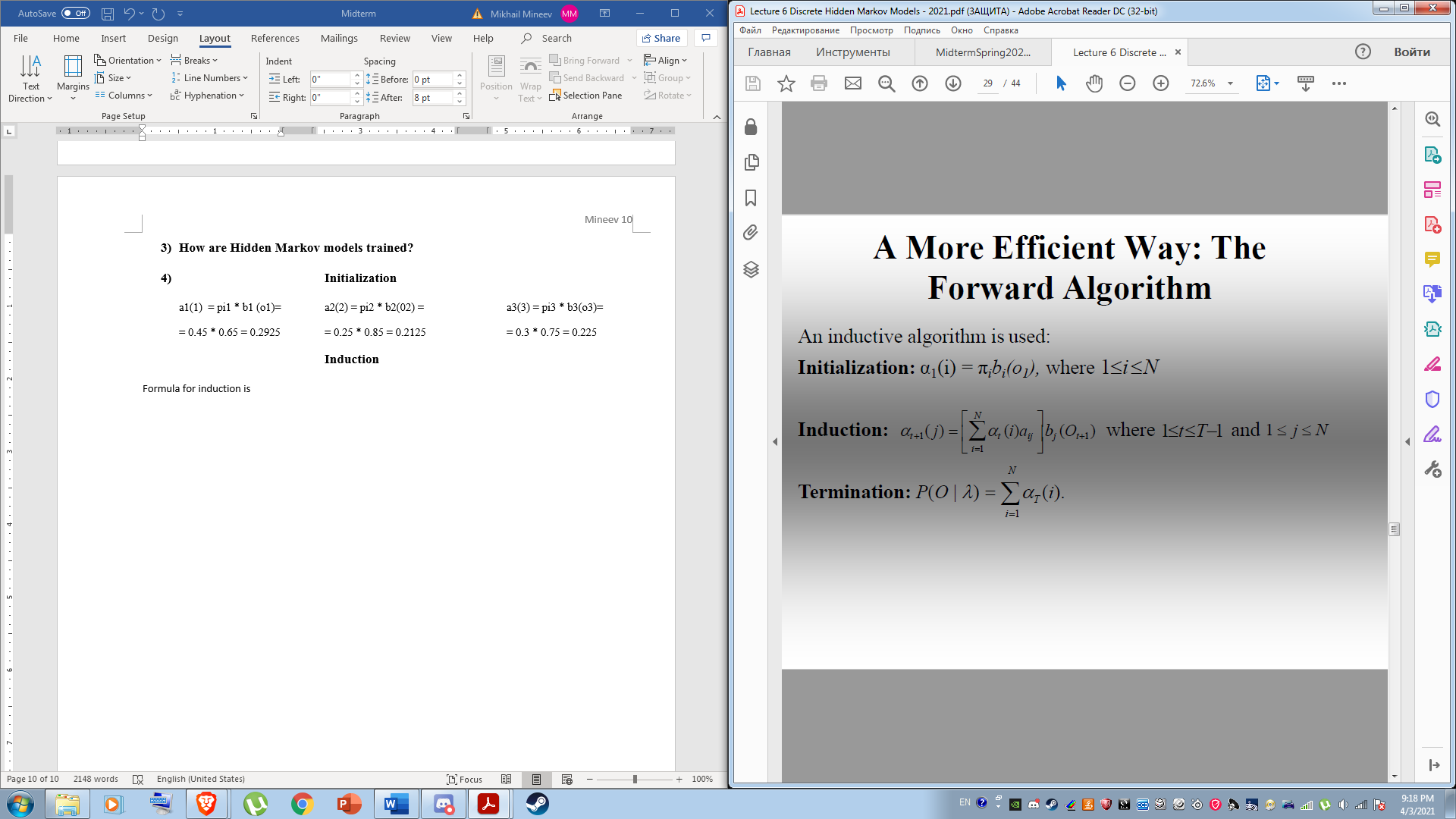
The implementation of this algorithm is largely dependent on understanding the main formulas being used. First it is necessary to initialize the initial layer of the array in order to allow the algorithm to run on the remaining layers. The last step, termination, is required to not only finish the run, but also to calculate the total probability of the algorithm going forward.

**Initialization**

alpha1(1) = pi1 \* b1 (o1)= alpha1(2) = pi2 \* b2(01) =

= 1.0 \* 0.7 = 0.7 = 0.0 \* 0.4 = 0.0

**Induction**

**Formula: **

**Row 1:**

alpha2(1) = [(0.7 \* 0.2) + (0.0 \* 0.6)] \* 0.3 = (0.14\* 0.3) = 0.042

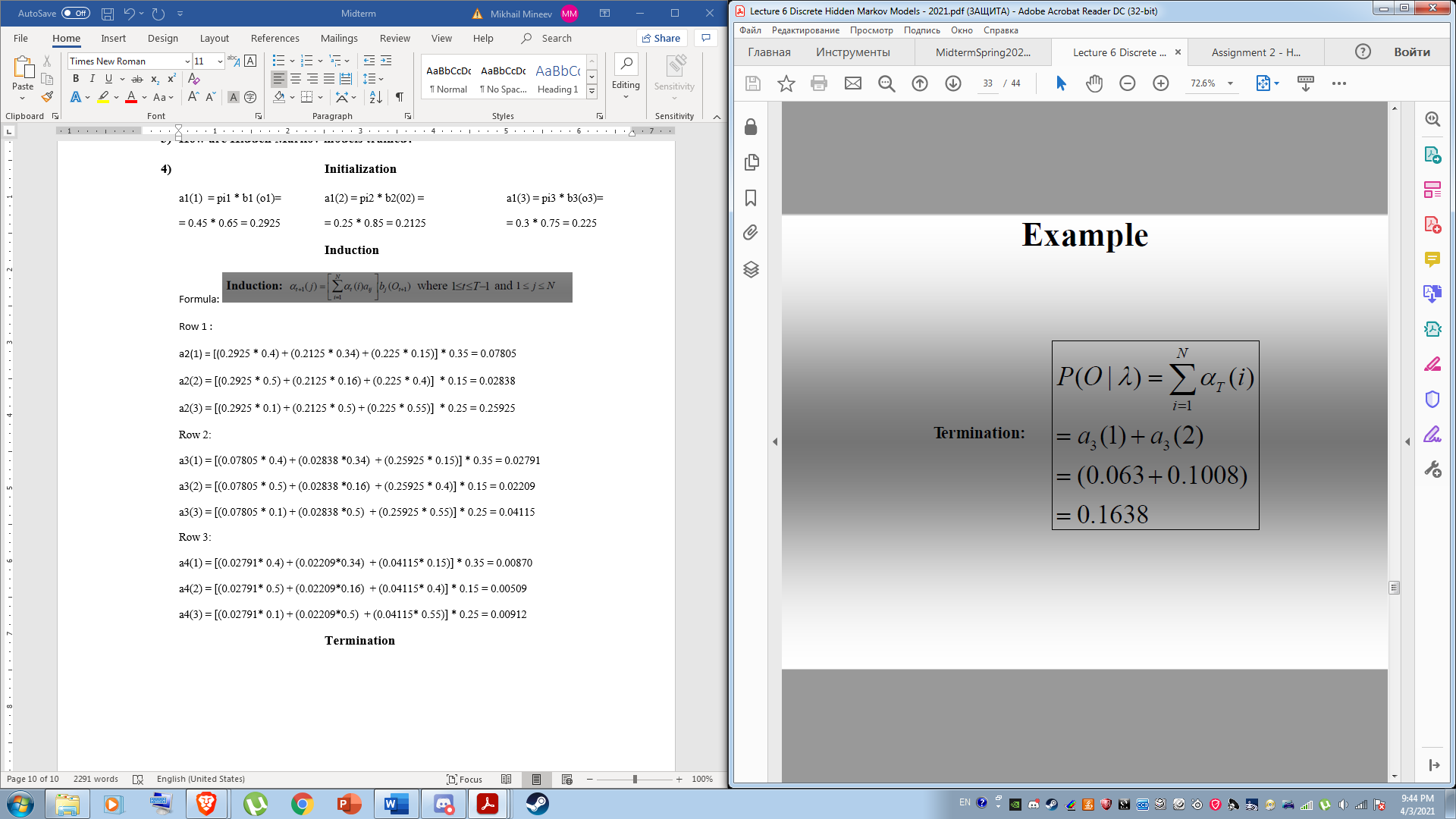
alpha2(2) = [(0.7 \* 0.8) + ( 0.0 \* 0.4)] \* 0.6 = (0.56 \* 0.6) = 0.336

**Row 2:**

alpha3(1) = [(0.042 \* 0.2) + (0.336 \* 0.6)] \* 0.3 = (0.21 \* 0.3) = 0.063

alpha3(2) = [(0.042 \* 0.8) + ( 0.336 \* 0.4)] \* 0.6 = (0.168 \* 0.6) = 0.1008

**Termination**

**Formula:** 

P(O | Lambda) = alpha3(1) + alpha 3(2) = 0.063+0.1008 = 0.1638

Formulas were taken from professor Konopka’s presentation 6 slide 29.

Part 2 result: Run HW2\_2\_2\_cs582\_Mikhail\_Mineev

**Initialization**

[[ 0.7000 0.0000]

[ 0.0000 0.0000]

[ 0.0000 0.0000]]

**Induction**

alpha( 2 , 1 )

[[ 0.7000 0.0000]

[ 0.0420 0.0000]

[ 0.0000 0.0000]]

alpha( 2 , 2 )

[[ 0.7000 0.0000]

[ 0.0420 0.3360]

[ 0.0000 0.0000]]

alpha( 3 , 1 )

[[ 0.7000 0.0000]

[ 0.0420 0.3360]

[ 0.0630 0.0000]]

alpha( 3 , 2 )

[[ 0.7000 0.0000]

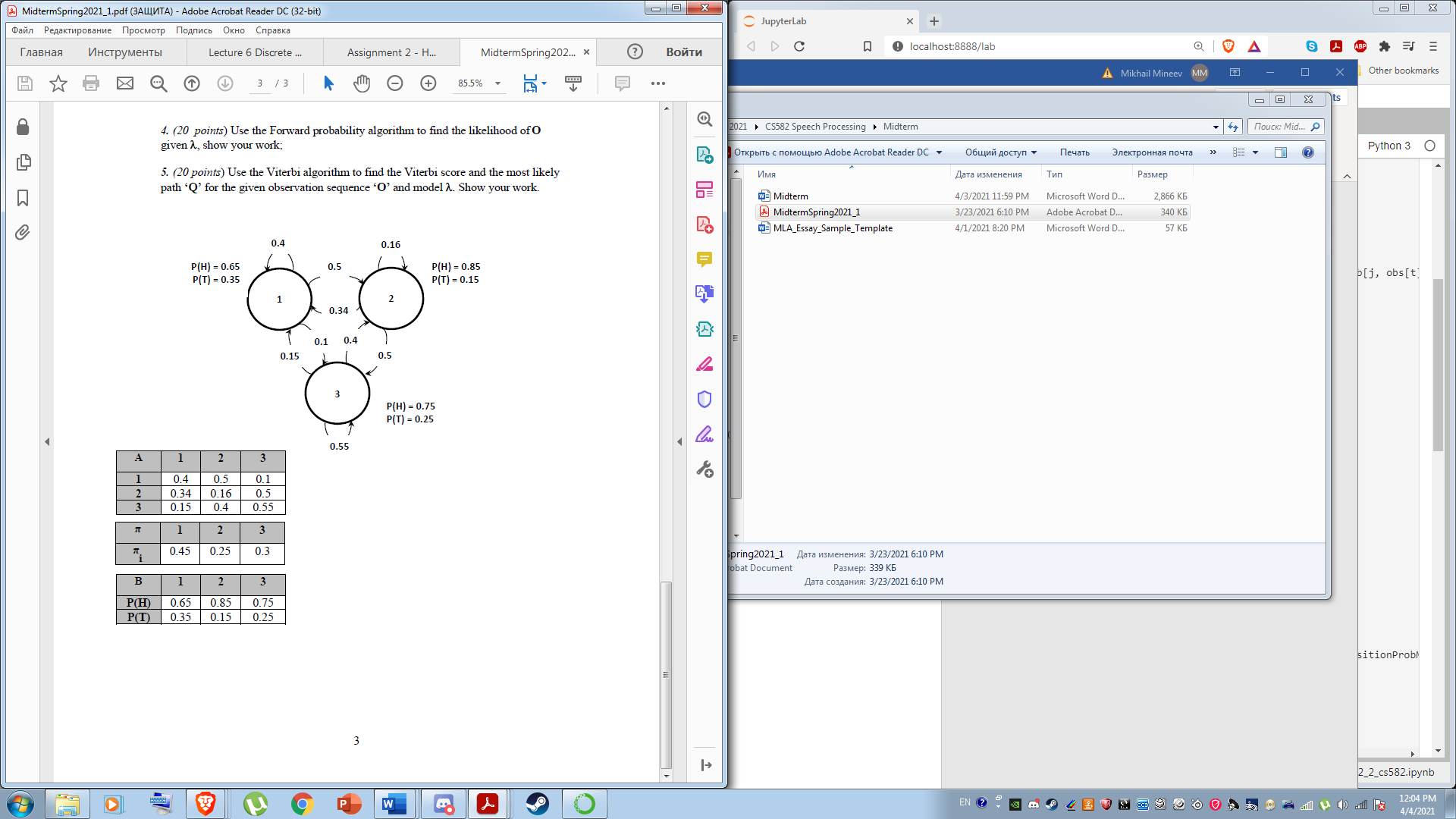
[ 0.0420 0.3360]

[ 0.0630 0.1008]]

**Termination**

P(O|λ)=0.1638

Part 3: Run HW2\_2\_3\_CS582\_Mikhail\_Mineev

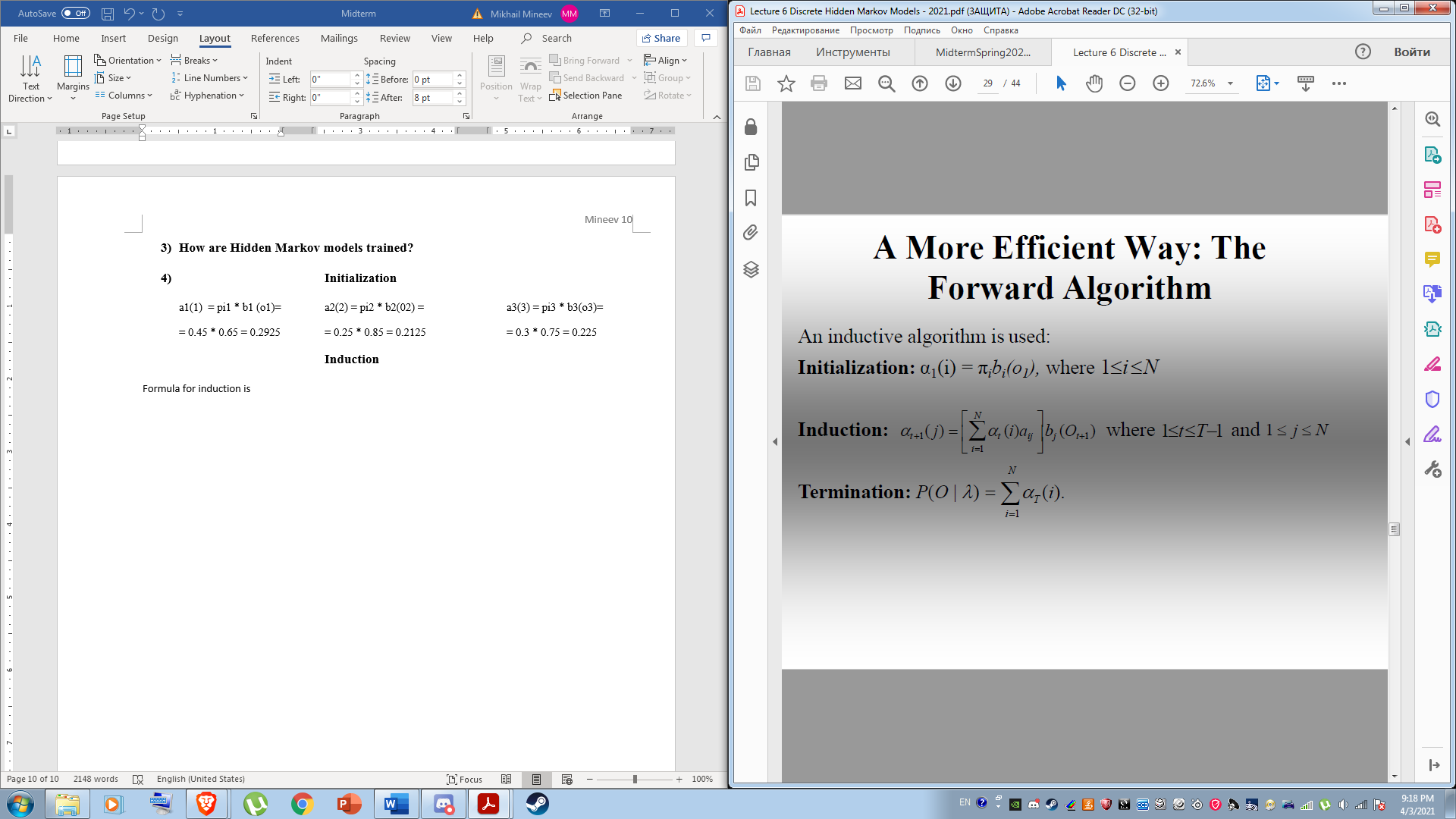


**Initialization**

alpha1(1) = pi1 \* b1 (o1)= alpha1(2) = pi2 \* b2(01) = alpha1(3) = pi3 \* b3(o1)=

= 0.45 \* 0.65 = 0.2925 = 0.25 \* 0.85 = 0.2125 = 0.3 \* 0.75 = 0.225

**Induction**

**Formula: **

**Row 1 :**

alpha2(1) = [(0.2925 \* 0.4) + (0.2125 \* 0.34) + (0.225 \* 0.15)] \* 0.35 = 0.07805

alpha2(2) = [(0.2925 \* 0.5) + (0.2125 \* 0.16) + (0.225 \* 0.4)] \* 0.15 = 0.04053

alpha2(3) = [(0.2925 \* 0.1) + (0.2125 \* 0.5) + (0.225 \* 0.55)] \* 0.25 = 0.06481

**Row 2:**

alpha3(1) = [(0.07805 \* 0.4) + (0.04053\*0.34) + (0.06481 \* 0.15)] \* 0.35 = 0.01915

alpha3(2) = [(0.07805 \* 0.5) + (0.04053\*0.16) + (0.06481\* 0.4)] \* 0.15 = 0.01071

alpha3(3) = [(0.07805 \* 0.1) + (0.04053\*0.5) + (0.06481\* 0.55)] \* 0.25 = 0.1593

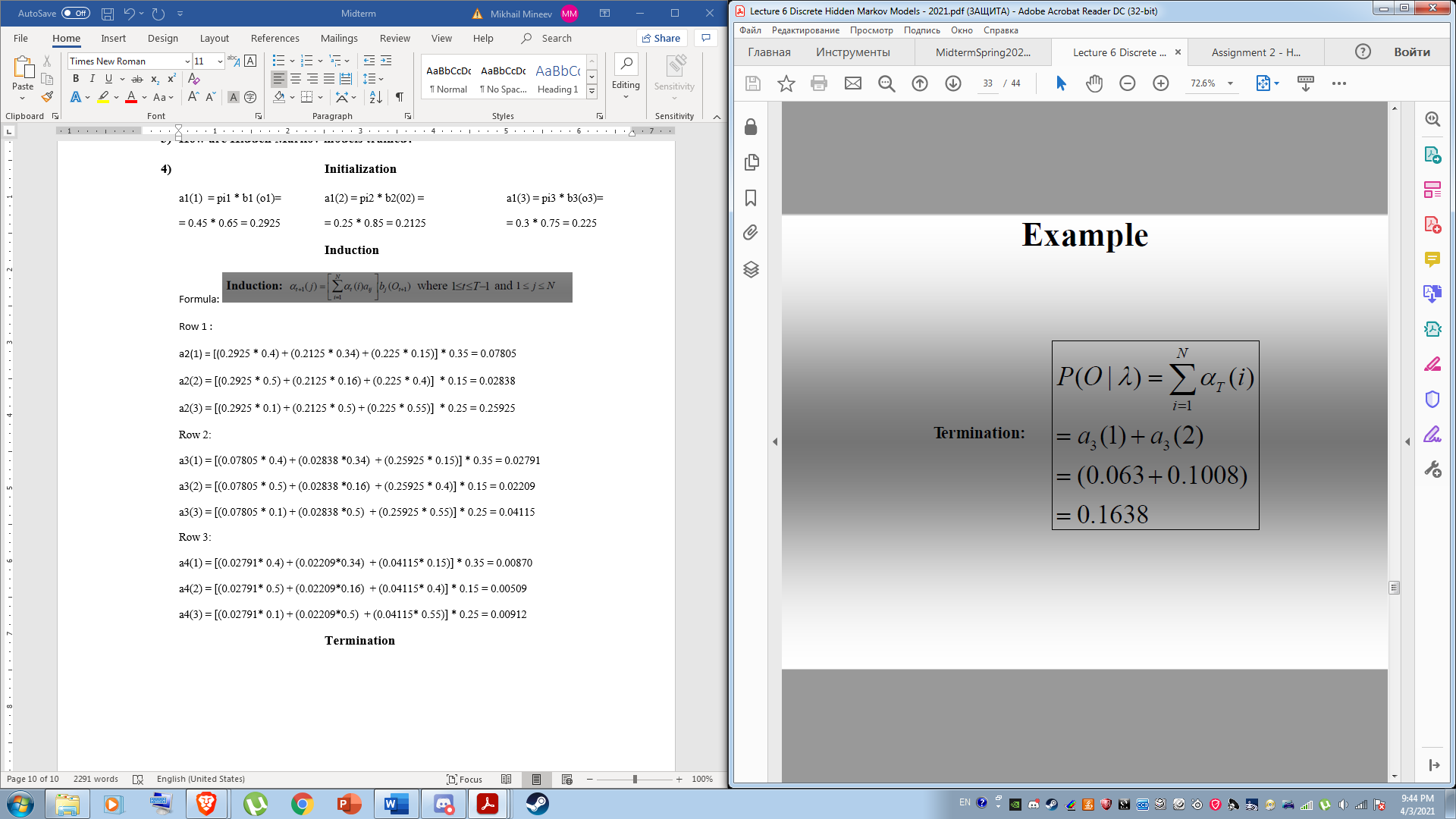
**Row 3:**

alpha4(1) = [(0.01915 \* 0.4) + (0.01071 \* 0.34) + (0.01593 \* 0.15)] \* 0.35 = 0.00479

alpha4(2) = [(0.01915 \* 0.5) + ( 0.01071 \*0.16) + (0.01593 \* 0.4)] \* 0.15 = 0.00265

alpha4(3) = [(0.01915 \* 0.1) + ( 0.01071 \*0.5) + (0.01593 \* 0.55)] \* 0.25 = 0.00401

**Termination**

**Formula:** 

P(O | lambda) = alpha4(1) + alpha4(2) + alpha4(3) = 0.00479+ 0.00265+ 0.00401= **0.01145**

**Run for part 3  
Initialization**

[[ 0.2925 0.2125 0.2250]

[ 0.0000 0.0000 0.0000]

[ 0.0000 0.0000 0.0000]

[ 0.0000 0.0000 0.0000]]

**Induction**

alpha( 2 , 1 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0000 0.0000]

[ 0.0000 0.0000 0.0000]

[ 0.0000 0.0000 0.0000]]

alpha( 2 , 2 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0405 0.0000]

[ 0.0000 0.0000 0.0000]

[ 0.0000 0.0000 0.0000]]

alpha( 2 , 3 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0405 0.0648]

[ 0.0000 0.0000 0.0000]

[ 0.0000 0.0000 0.0000]]

alpha( 3 , 1 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0405 0.0648]

[ 0.0192 0.0000 0.0000]

[ 0.0000 0.0000 0.0000]]

alpha( 3 , 2 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0405 0.0648]

[ 0.0192 0.0107 0.0000]

[ 0.0000 0.0000 0.0000]]

alpha( 3 , 3 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0405 0.0648]

[ 0.0192 0.0107 0.0159]

[ 0.0000 0.0000 0.0000]]

alpha( 4 , 1 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0405 0.0648]

[ 0.0192 0.0107 0.0159]

[ 0.0048 0.0000 0.0000]]

alpha( 4 , 2 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0405 0.0648]

[ 0.0192 0.0107 0.0159]

[ 0.0048 0.0026 0.0000]]

alpha( 4 , 3 )

[[ 0.2925 0.2125 0.2250]

[ 0.0781 0.0405 0.0648]

[ 0.0192 0.0107 0.0159]

[ 0.0048 0.0026 0.0040]]

**Termination**

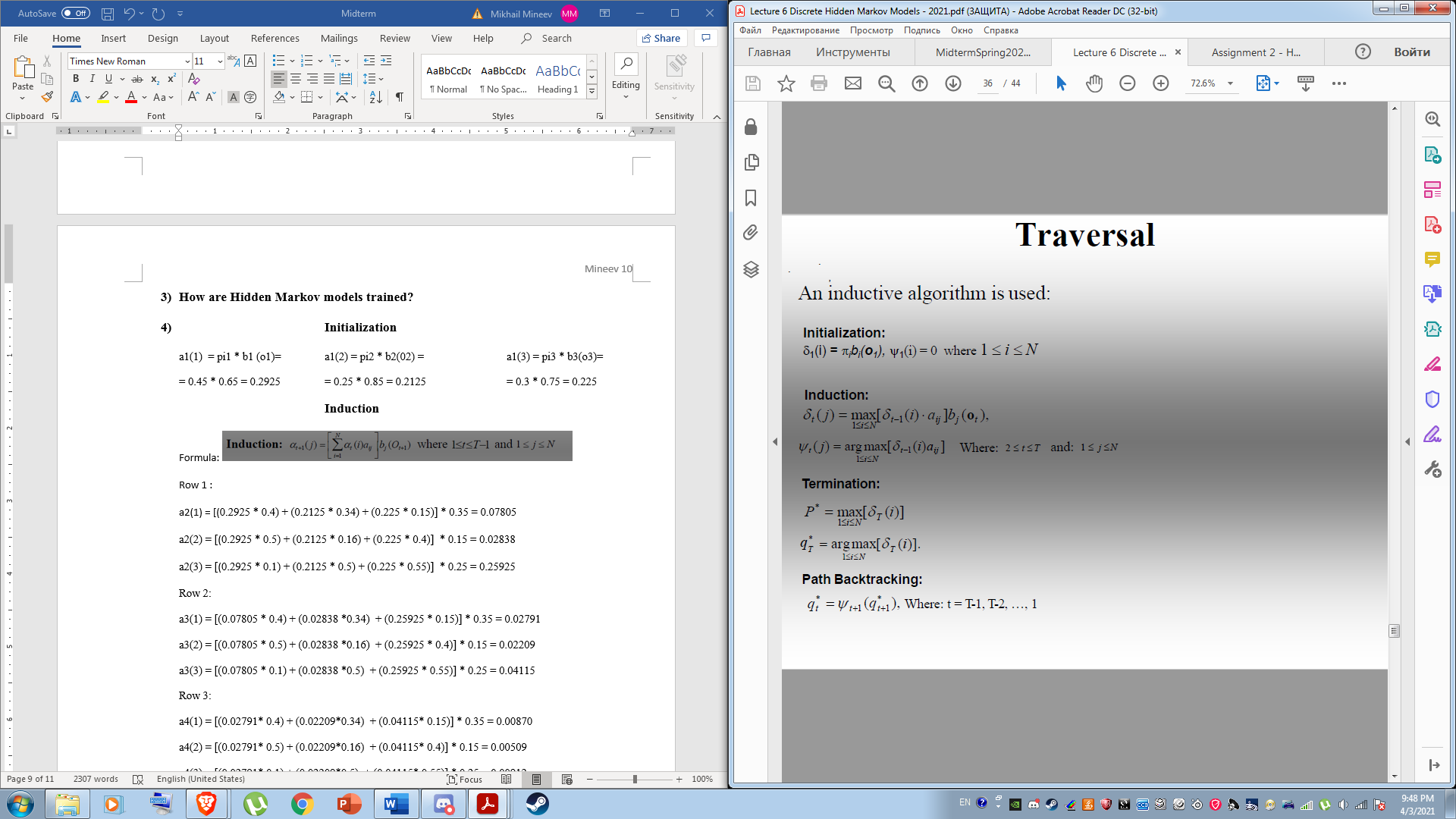
0.00801732390625

P(O|λ)=0.0115

1. **Viterby Algorithm**

This algorithm is designed to find the most probably path through the HMM. This algorithm has a few new variables introduced to it, one main one is the best path variable, into it we put the best overall path of the algorithm. To do this we also create a new variable, phi. Phi is responsible for finding the most probably path amongst the given paths using argmax. The following example will demonstrate this.

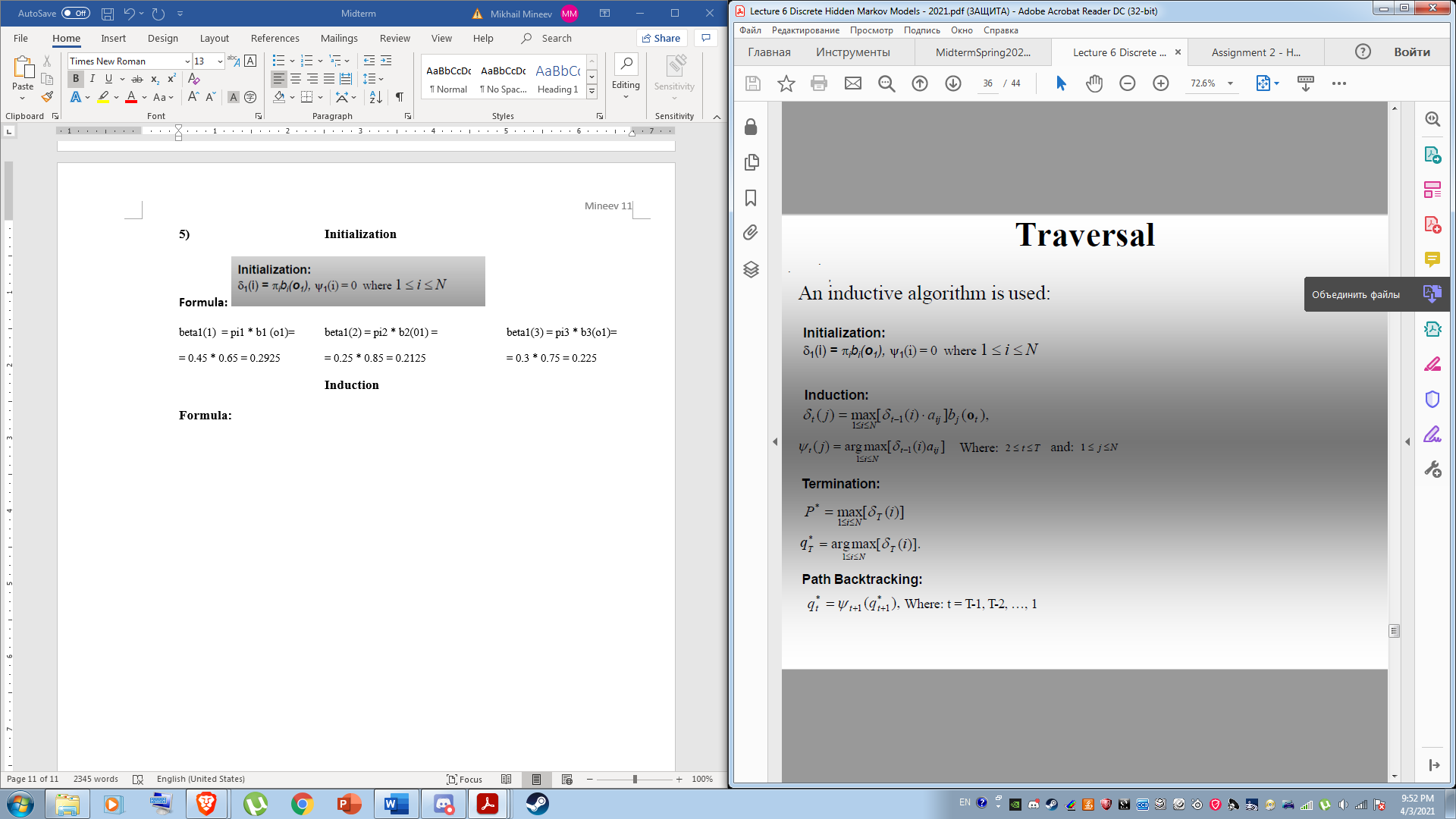
**Initialization**

**Formula:**

beta1(1) = pi1 \* b1 (o1)= beta1(2) = pi2 \* b2(01) =

= 1.0 \* 0.7 = 0.7 = 0.0 \* 0.4 = 0.0

**Induction**

**Formula:** 

**Row1:**

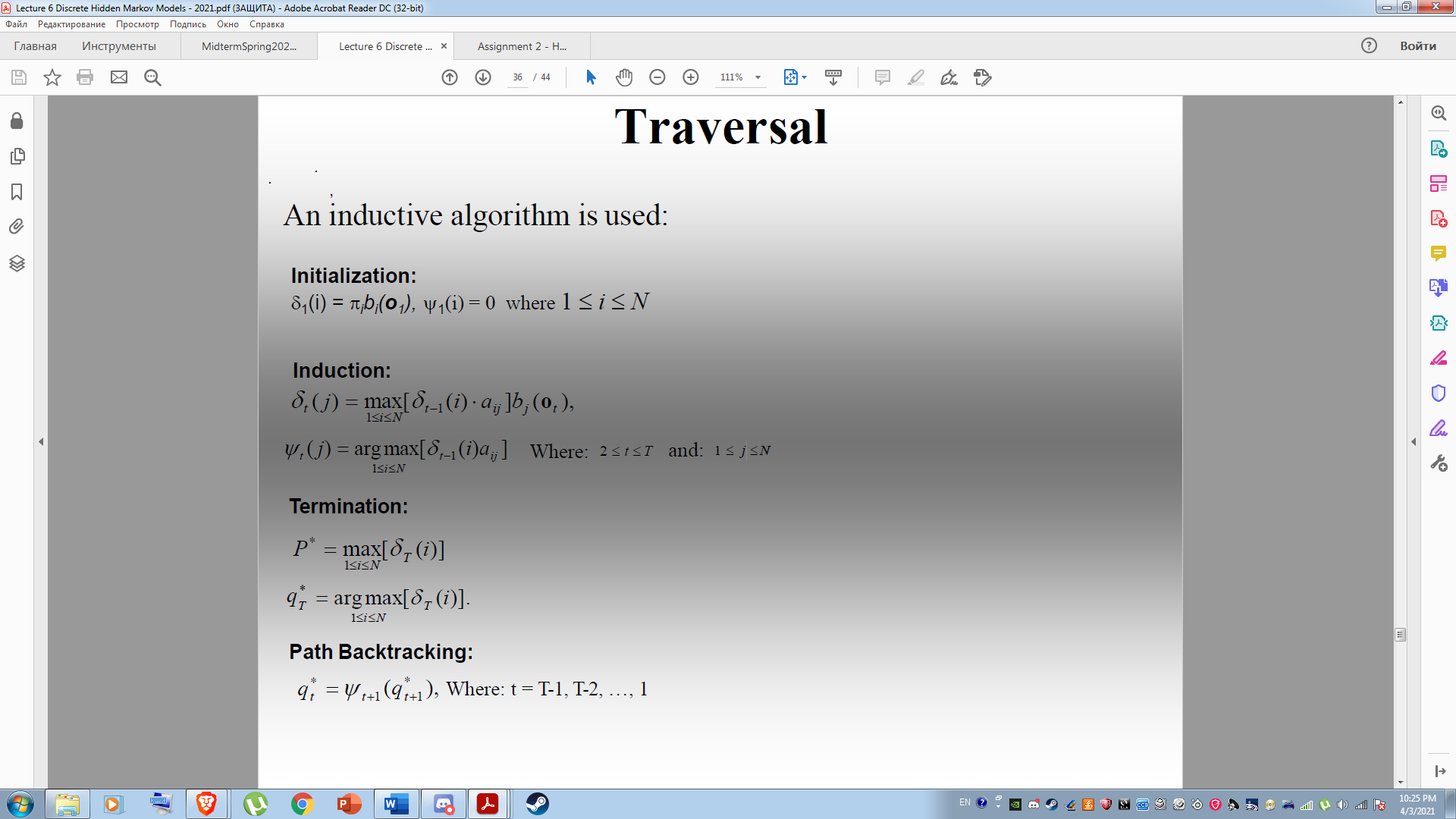
**beta2(1)** = max[(0.7 \* 0.2), (0.0 \* 0.6)] \* 0.3 = max[(0.14), (0.0)]\*0.35 = (0.14 \*0.35) = **0.042**

**phi2(1)** = argmax[0.14,0.0], I = 1

**beta2(2)** = max[(0.7 \* 0.8), (0.0 \* 0.4)] \* 0.6 = max[(0.56), (0.0)] \* 0.6 = (0.56 \*0.6) = **0.336**

**phi2(2) =** argmax[0.56, 0.0], I = 1

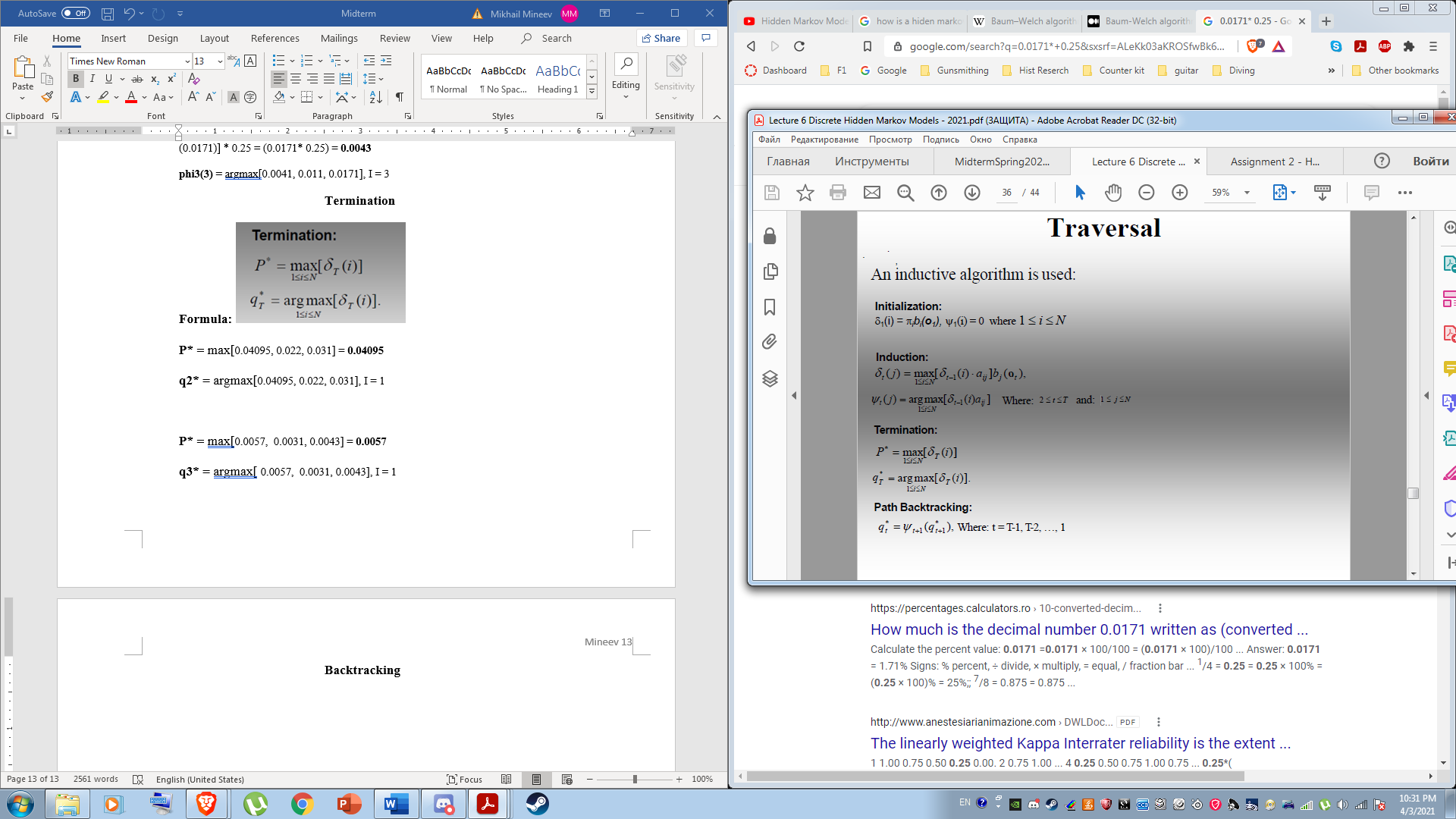
**Termination**

**Formula: **

**P\* =** max[0.049, 0.336] = **0.336**

**q2\* =** argmax[0.049, 0.336], I = 2

**Backtracking**

**Formula:**

**Best path for q1,q2 = 1, 2**

Program Output:

**Initialization**

**delta**

[[ 0.7000 0.0000]

[ 0.0000 0.0000]

[ 0.0000 0.0000]]

**psi**

[[0. 0.]

[0. 0.]

[0. 0.]]

**Induction**

Processed delta( 2 , 1 )

[[ 0.7000000000 0.0000000000]

[ 0.0420000000 0.0000000000]

[ 0.0000000000 0.0000000000]]

Processed psi( 2 , 1 )

[[0. 0.]

[1. 0.]

[0. 0.]]

Processed delta( 2 , 2 )

[[ 0.7000000000 0.0000000000]

[ 0.0420000000 0.3360000000]

[ 0.0000000000 0.0000000000]]

Processed psi( 2 , 2 )

[[0. 0.]

[1. 1.]

[0. 0.]]

Processed delta( 3 , 1 )

[[ 0.7000000000 0.0000000000]

[ 0.0420000000 0.3360000000]

[ 0.0604800000 0.0000000000]]

Processed psi( 3 , 1 )

[[0. 0.]

[1. 1.]

[2. 0.]]

Processed delta( 3 , 2 )

[[ 0.7000000000 0.0000000000]

[ 0.0420000000 0.3360000000]

[ 0.0604800000 0.0806400000]]

Processed psi( 3 , 2 )

[[0. 0.]

[1. 1.]

[2. 2.]]

**Viterbi Score**

0.08063999999999999

**Best Path**

[1 2 2]