REPORT

KMU 국민대학교 KOOKMIN UNIVERSITY

과목명 | 빅데이터최신기술

담당교수 |

학과 |

학년 |

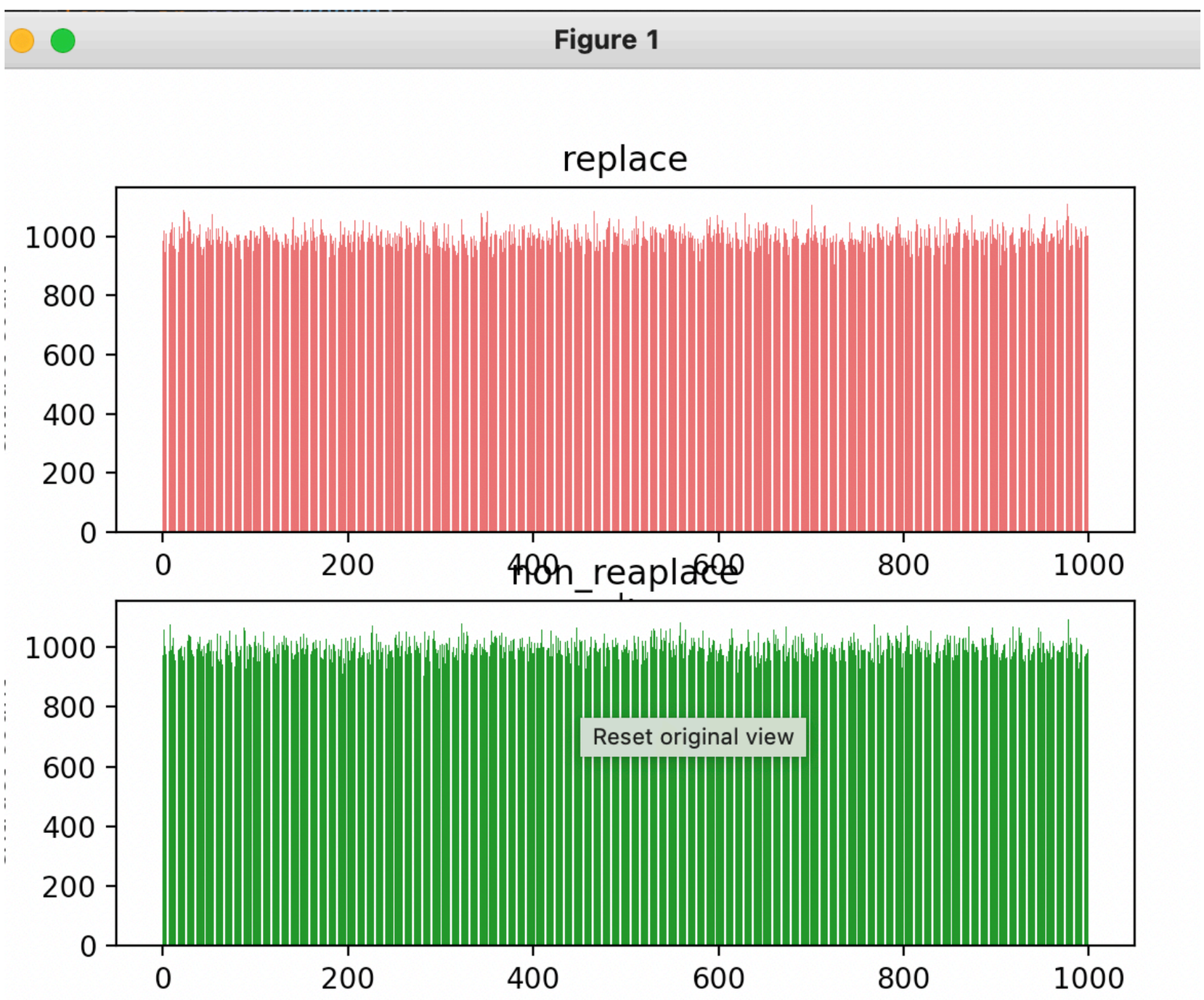
학번 | 20181703

이름 | 평선호

제출일 |

Reservoir Sampling / DGIM Algorithm 보고서

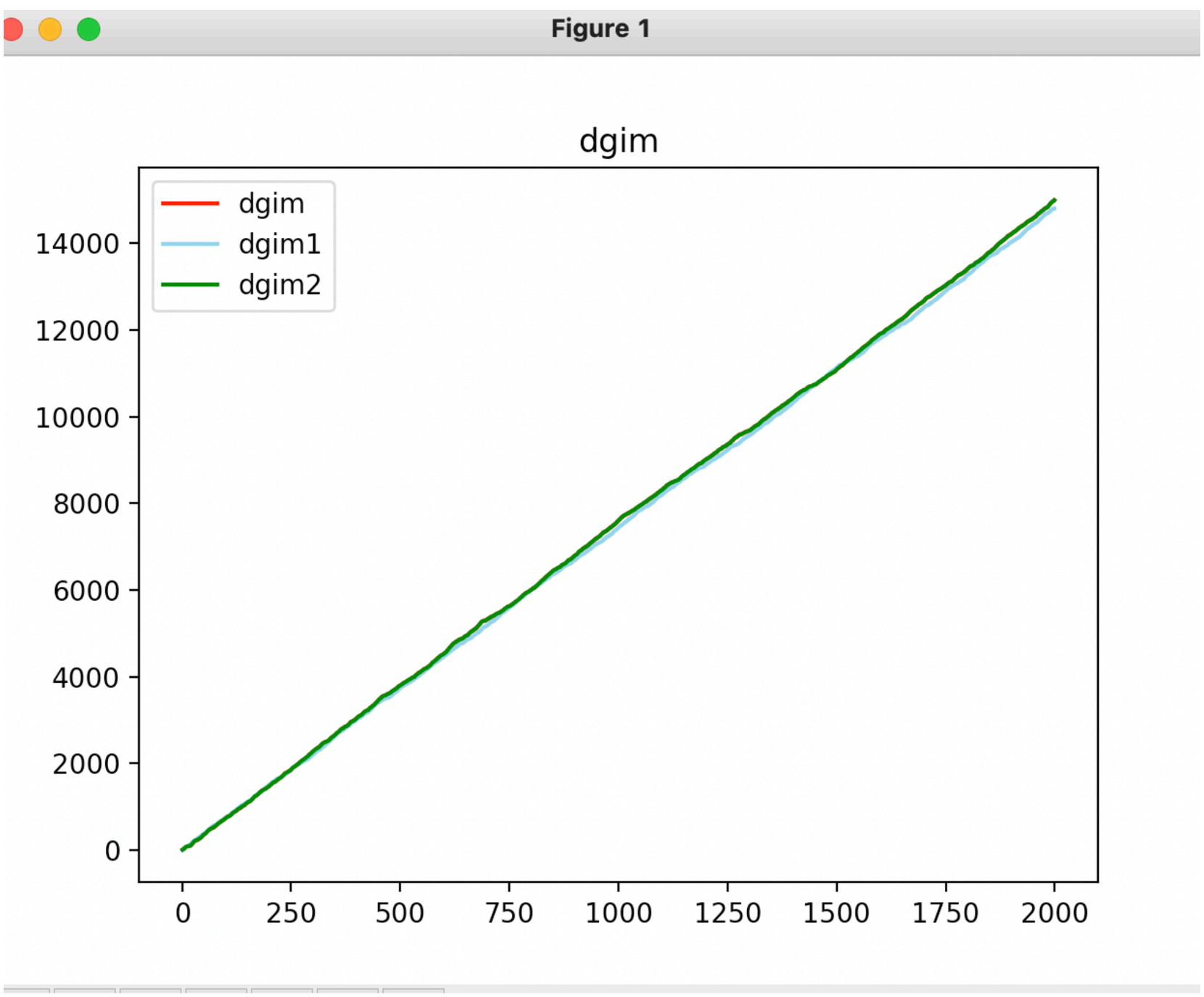
1. Reservoir Sampling



위에서부터 복원추출, 비복원추출을 한 결과물

비복원 추출과 복원추출 둘다 stream이 1에서 1000까지 들어왔을 때 100번 추출을 했을 때 결과물입니다. 이때 비복원추출은 k를 100으로 설정하고 복원추출은 k가 1을 100번 반복합니다. 이때 스트림이 추출된 값은 대략 1000에 근접해서 균등하게 나옵니다.

2. DGIM Algorithm



Dgim 기본 값과 응용 dgim 1번 방법과 2번 방법으로 구한 그래프 값을 나타낸것입니다.

Dgim()기본 알고리즘

```
class Dgim():
    def __init__(self):
        self.bucket_tower = [[]]
        self.ts = 0
    def put(self,bits):

        if bits == 1:
            self.bucket_tower[0].insert(0,Bucket(self.ts,self.ts))

            layer = 0
            if len(self.bucket_tower) < 2:
                while len(self.bucket_tower[layer]) > 2:
                    if len(self.bucket_tower) <= layer + 1:
                        self.bucket_tower.append([])

                    b1 = self.bucket_tower[layer].pop()
                    b2 = self.bucket_tower[layer].pop()
                    b1.end = b2.end

                    self.bucket_tower[layer+1].insert(0,b1)
                    layer +=1
        self.ts +=1
```

```
def count(self,k):
    s = self.ts - k
    cnt = 0

    for layer,buckets in enumerate(self.bucket_tower):
        for bucket in buckets:
            if s <= bucket.start:
                cnt += (1 << layer)
            elif s <= bucket.end:
                cnt += (1 << layer) * (bucket.end-s +1) // (bucket.end - bucket.start +1)
                return cnt
            else:
                return cnt
    return cnt
```

Dgim ()응용1 알고리즘

```
dgim1_layer = [[],[],[],[]]
bit_0_stream = []
bit_1_stream = []
bit_2_stream = []
bit_3_stream = []
bit = []
```

```
    a = (random.randint(0,15))
    str = format(a, 'b')
    str = str.zfill(4)
    for i in range(len(str)):
        if i == 0:
            bit_3_stream.append(int(str[i]))
        elif i == 1:
            bit_2_stream.append(int(str[i]))
        elif i == 2:
            bit_1_stream.append(int(str[i]))
        elif i == 3:
            bit_0_stream.append(int(str[i]))
```

```
real_answer = []
dgim1_answer = []
for k in range(1,2000):
    real_answer.append(sum(bitstream[-k:]))
    total_dgim1 = dgim1_layer[0][k-1] * 1 + dgim1_layer[1][k-1] * 2 + dgim1_layer[2][k-1] * 4 + dgim1_layer[3][k-1] * 8
    dgim1_answer.append(total_dgim1)
```

기본적으로 dgim()1 응용은 bit stream을 바탕으로 돌아가는 알고리즘이기 때문에
기존 dgim()알고리즘과 같은 함수를 사용했습니다.

초기 정수값을 binary형태로 변환해주며 zfill(4)로 이용하여 4비트 형태로 바꿔줍니다.
이후 해당 비트별로 bit_stream에 추가해줍니다.
이후 각 bit_stream에 있는 0과 1의 비트들을 dgim()알고리즘을 이용하여 진행해줍니다.

digm1_layer는 dgim()을 이용하고 나온 값들을 넣어주는 변수 입니다.

Dgim ()응용2 알고리즘

```
def put(self,bits):
    self.bucket_tower[0].insert(0,bits)
    layer = 0
    while len(self.bucket_tower[layer]) > 2:
        if len(self.bucket_tower) < layer + 2:
            self.bucket_tower.append([])
        if type(self.bucket_tower[layer][-1]) is list:# layer[-1]이 list일 때
            if type(self.bucket_tower[layer][-2]) is list:# layer[-2]이 list일 때
                if sum(self.bucket_tower[layer][-1]) + sum(self.bucket_tower[layer][-2]) <= (2**(layer+1)):# 2^b보다 작을때 2개를 올림
                    a1 = self.bucket_tower[layer].pop()
                    a2 = self.bucket_tower[layer].pop()
                    self.bucket_tower[layer + 1].insert(0, a1)
                    self.bucket_tower[layer + 1].insert(0, a2)
                else:  # 2^b 보다 클때 1개만 올림
                    a1 = self.bucket_tower[layer].pop()
                    self.bucket_tower[layer + 1].insert(0, a1)

            elif type(self.bucket_tower[layer][-2]) is int:# layer[-2]이 int일 때
                if sum(self.bucket_tower[layer][-1]) + self.bucket_tower[layer][-2] <= (2 ** (layer + 1)):  # 2^b보다 작을때 2개를 올림
                    a1 = self.bucket_tower[layer].pop()
                    a2 = self.bucket_tower[layer].pop()
                    a1.insert(0,a2)
                    self.bucket_tower[layer + 1].insert(0, a1)
                else:  # 2^b 보다 클때 1개만 올림
                    a1 = self.bucket_tower[layer].pop()
                    self.bucket_tower[layer + 1].insert(0, a1)
```

```
        elif type(self.bucket_tower[layer][-1]) is int:# layer[-1]이 int일 때
            if type(self.bucket_tower[layer][-2]) is list:
                if self.bucket_tower[layer][-1] + sum(self.bucket_tower[layer][-2]) <= (2**(layer+1)):# 2^b보다 작을때 2개를 올림
                    a1 = self.bucket_tower[layer].pop()
                    a2 = self.bucket_tower[layer].pop()
                    a2.append(a1)
                    self.bucket_tower[layer + 1].insert(0, a2)
                else:  # 2^b 보다 클때 1개만 올림
                    a1 = self.bucket_tower[layer].pop()

                    self.bucket_tower[layer + 1].insert(0, a1)

            elif type(self.bucket_tower[layer][-2]) is int:
                if self.bucket_tower[layer][-1] + self.bucket_tower[layer][-2] <= (2**(layer+1)):# 2^b보다 작을때 2개를 올림
                    a1 = self.bucket_tower[layer].pop()
                    a2 = self.bucket_tower[layer].pop()
                    self.bucket_tower[layer + 1].insert(0, [a1, a2])
                else:  # 2^b 보다 클때 1개만 올림
                    a1 = self.bucket_tower[layer].pop()
                    self.bucket_tower[layer + 1].insert(0, a1)
        layer +=1
```

각 층의 layer들은 3개이상이 되면 안되고 3개 이상이 됐을 때 먼저 들어온 값들을 더해서 2^b보다 작으면 2개를 올려주고 아니면 1개만 올려줍니다.

이 때 2개가 올라가게 되면 해당 숫자들을 묶어서 올려야 하므로 집합을 이용했습니다. 이때 그이후에 숫자를 더할 때 집합과 숫자를 더하는 경우가 생기는데 집합과 숫자를 더할 수 없으므로 해당 조건들을 조건문을 이용하여 나눠서 진행합니다.

```
idx = [0]
summ = [0]
b = 0
id = 0
for k in range(len(dgim2.bucket_tower)):
    for i in range(len(dgim2.bucket_tower[k])):
        if type(dgim2.bucket_tower[k][i]) is int:
            b +=1
            idx.append(b)
            c = summ[-1]
            c = c + dgim2.bucket_tower[k][i]
            summ.append(c)
        elif type(dgim2.bucket_tower[k][i]) is list:
            if len(dgim2.bucket_tower[k]) == 1:
                idx.append(b +len(dgim2.bucket_tower[k][i]))
                b += len(dgim2.bucket_tower[k][i])
                c = summ[-1]
                c = c + sum(dgim2.bucket_tower[k][i])
                summ.append(c)
                continue
            elif len(dgim2.bucket_tower[k]) == 2:
                idx.append(b + len(dgim2.bucket_tower[k][i]))
                b += len(dgim2.bucket_tower[k][i])
                c = summ[-1]
                c = c + sum(dgim2.bucket_tower[k][i])
                summ.append(c)
```

```
dgim2_answer = [0]
id = 1
for k in range(1,2000):
    if idx[id] == k:
        dgim2_answer.append(summ[id])
        id +=1
    else:
        next = idx[id]
        before = idx[id-1]
        next_sum = summ[id]
        before_sum = summ[id-1]
        rest = next-before
        rest_sum = next_sum - before_sum
        rest_sum = (1- (idx[id]-k)/rest) * rest_sum
        dgim2_answer.append((round(summ[id-1] + rest_sum)))
```

dgim2()를 이용하여 만들어진 bucket_tower에 있는 정수값을 이제 계산해줍니다. 각 인덱스에 있는 bucket_tower에 있는 길이는 2이므로 ex) a[k][i]로 접근을 합니다. 이때 접근을 했을 때 bucket_tower안에 있는 값들은 list일 경우가 있고 int가 있는 경우가 있는데 이때 조건문을 이용하여 계산해줍니다.

```
for k in range(1,2000):
    real_answer.append(sum(bitstream[-k:]))
    total_dgim1 = dgim1_layer[0][k-1] * 1 + dgim1_layer[1][k-1] * 2 + dgim1_layer[2][k-1] * 4 + dgim1_layer[3][k-1] * 8
    dgim1_answer.append(total_dgim1)



plt.plot([i for i in range(1,2000)],real_answer,color='red',label='dgim')
plt.plot([i for i in range(1,2000)],dgim1_answer,color='skyblue',label='dgim1')
plt.plot([i for i in range(1,2000)],dgim2_answer[1:],color='green',label='dgim2')
plt.title("dgim")
plt.legend(loc='upper left')
plt.show()
```

```python
ddd = dgim2_answer[1:]
for i in range(1,2000,20):
    print(real_answer[i], dgim1_answer[i], ddd[i])
```

해당 결과물들의 값들이 어떻게 진행되는지 알고자 20으로 나눠서 출력합니다.

```
1740 1692 1740
1897 1875 1898
2036 2025 2036
2172 2147 2172
2302 2288 2299
2394 2405 2402
2526 2573 2526
2675 2723 2675
2807 2912 2807
2960 3069 2960
3096 3166 3096
3250 3287 3249
3406 3461 3401
3574 3592 3572
3733 3754 3733
3866 3955 3866
4023 4093 4022
4184 4200 4182
4340 4359 4342
4457 4509 4456
4587 4647 4587
4729 4842 4729
4856 4970 4863
5000 5116 4999
5175 5250 5176
5341 5383 5341
5455 5543 5455
5590 5706 5590
5724 5853 5724
5909 6028 5909
6057 6157 6057
```

```
6215 6333 6215
6392 6516 6388
6535 6683 6540
6724 6825 6724
6902 6992 6902
7112 7133 7112
7287 7298 7286
7447 7458 7444
7578 7581 7573
7699 7768 7704
7841 7944 7842
7980 8097 7985
8132 8258 8136
8308 8403 8308
8483 8562 8483
8652 8713 8652
8836 8887 8836
8984 9075 8982
9124 9268 9122
9247 9406 9243
9363 9560 9370
9506 9687 9512
9656 9846 9658
9801 10013 9800
9930 10172 9930
10053 10363 10053
10178 10511 10178
10297 10692 10301
10486 10839 10484
```

```
10617 10964 10617
10796 11104 10796
10924 11259 10919
11087 11430 11088
11277 11576 11277
11462 11690 11466
11611 11886 11611
11750 12025 11744
11892 12159 11894
12061 12307 12061
12207 12433 12207
12383 12577 12382
12526 12741 12528
12663 12870 12663
12789 12993 12789
12958 13106 12958
13082 13254 13085
13234 13434 13234
13387 13614 13387
13500 13742 13500
13641 13887 13643
13852 13988 13852
14007 14163 14007
14167 14322 14167
14301 14453 14304
14463 14596 14460
14594 14737 14592
14734 14882 14736
14883 15023 14883
```

결과 값을 보면 거의 real_answer과 dgim2가 비슷한 값으로 진행하는 모습을 볼수 있습니다.
따라서 dgim2가 dgim1보다 좀더 정확도가 높다고 생각했습니다.