Diagonalize

Andrea Mazzotti

<amm263@gmail.com>

Index

Given Problem

Input:

- **V** and **W** ordered arrays of numbers.
- ***k*** number

Find ***k*** as sum of **V[v]** and **W[w]**.

If **V[v]+W[w]** = ***k*** then return *v* and *w*

Note:
I don't know the name of this problem and the name of the known algorithm to solve it either.
I tried to find the name of the algorithm, but no one seems to know it. The best answer received was: *"Maybe this algorithm doesn't have a name, because it is too stupid to deserve one"*.
I called the algorithm Walker, because it simply...walks.

Walker: a known linear algorithm

Working Example

The square matrix is the result of V[v]+W[w] for each v and w.
I chose a square matrix (v = w) for simplicity.

1) k = 148

	V	1	12	33	35	67	70	75	89	112	120	210
W												
2		3	14	35	37	69	72	77	91	114	122	212
10		11	22	43	45	77	80	85	99	122	130	220
18		19	30	51	53	85	88	93	107	130	138	228
22		23	34	55	57	89	92	97	111	134	142	232
34		35	46	67	69	101	104	109	123	146	154	244
55		56	67	88	90	122	125	130	144	167	175	265
78		79	90	111	113	145	148	153	167	190	198	288
87		88	99	120	122	154	157	162	176	199	207	297
99		100	111	132	134	166	169	174	188	211	219	309
156		157	168	189	191	223	226	231	245	268	276	366
170		171	182	203	205	237	240	245	259	282	290	380

2) The starting cell is always W[w]+V[0].

	V	1	12	33	35	67	70	75	89	112	120	210
W												
2		3	14	35	37	69	72	77	91	114	122	212
10		11	22	43	45	77	80	85	99	122	130	220
18		19	30	51	53	85	88	93	107	130	138	228
22		23	34	55	57	89	92	97	111	134	142	232
34		35	46	67	69	101	104	109	123	146	154	244
55		56	67	88	90	122	125	130	144	167	175	265
78		79	90	111	113	145	148	153	167	190	198	288
87		88	99	120	122	154	157	162	176	199	207	297
99		100	111	132	134	166	169	174	188	211	219	309
156		157	168	189	191	223	226	231	245	268	276	366
170		171	182	203	205	237	240	245	259	282	290	380

3) Now the rules are simple:

If we find a number higher than k, we need to move up, else we need to move right. Let's see.

	V	1	12	33	35	67	70	75	89	112	120	210
W												
2		3	14	35	37	69	72	77	91	114	122	212
10		11	22	43	45	77	80	85	99	122	130	220
18		19	30	51	53	85	88	93	107	130	138	228
22		23	34	55	57	89	92	97	111	134	142	232
34		35	46	67	69	101	104	109	123	146	154	244
55		56	67	88	90	122	125	130	144	167	175	265
78		79	90	111	113	145	148	153	167	190	198	288
87		88	99	120	122	154	157	162	176	199	207	297
99		100	111	132	134	166	169	174	188	211	219	309
156		157	168	189	191	223	226	231	245	268	276	366
170		171	182	203	205	237	240	245	259	282	290	380

4) 157 > k (148) then move up again

	V	1	12	33	35	67	70	75	89	112	120	210
W												
2		3	14	35	37	69	72	77	91	114	122	212
10		11	22	43	45	77	80	85	99	122	130	220
18		19	30	51	53	85	88	93	107	130	138	228
22		23	34	55	57	89	92	97	111	134	142	232
34		35	46	67	69	101	104	109	123	146	154	244
55		56	67	88	90	122	125	130	144	167	175	265
78		79	90	111	113	145	148	153	167	190	198	288
87		88	99	120	122	154	157	162	176	199	207	297
99		100	111	132	134	166	169	174	188	211	219	309
156		157	168	189	191	223	226	231	245	268	276	366
170		171	182	203	205	237	240	245	259	282	290	380

5) 100 < k (148) then move right

	V	1	12	33	35	67	70	75	89	112	120	210
W												
2		3	14	35	37	69	72	77	91	114	122	212
10		11	22	43	45	77	80	85	99	122	130	220
18		19	30	51	53	85	88	93	107	130	138	228
22		23	34	55	57	89	92	97	111	134	142	232
34		35	46	67	69	101	104	109	123	146	154	244
55		56	67	88	90	122	125	130	144	167	175	265
78		79	90	111	113	145	148	153	167	190	198	288
87		88	99	120	122	154	157	162	176	199	207	297
99		100	111	132	134	166	169	174	188	211	219	309
156		157	168	189	191	223	226	231	245	268	276	366
170		171	182	203	205	237	240	245	259	282	290	380

6) After 10 steps we will find our k

W \ V	1	12	33	35	67	70	75	89	112	120	210
2	3	14	35	37	69	72	77	91	114	122	212
10	11	22	43	45	77	80	85	99	122	130	220
18	19	30	51	53	85	88	93	107	130	138	228
22	23	34	55	57	89	92	97	111	134	142	232
34	35	46	67	69	101	104	109	123	146	154	244
55	56	67	88	90	122	125	130	144	167	175	265
78	79	90	111	113	145	148	153	167	190	198	288
87	88	99	120	122	154	157	162	176	199	207	297
99	100	111	132	134	166	169	174	188	211	219	309
156	157	168	189	191	223	226	231	245	268	276	366
170	171	182	203	205	237	240	245	259	282	290	380

Pseudo code

```
i = 0;
j = w;

while( i<v AND j<w)
{
        if V[i]+W[j] > k
                w-- ;
        else if V[i]+W[j] < k
                i++ ;
        else return i,j;
}
k not found;
```

Computational Cost

The cost is linear and in the worst case, considering a square matrix $n \times n$, is 2n for the Manhattan distance principle.
The worst case is when k is not in the matrix.

Diagonalize: a better solution

Working Example

k = 114

1) We start considering the diagonal of the square matrix and executing a binary search on it, to find k.

	V	1	12	33	35	67	70	75	89	112	120
W											
2		3	14	35	37	69	72	77	91	114	122
10		11	22	43	45	77	80	85	99	122	130
18		19	30	51	53	85	88	93	107	130	138
22		23	34	55	57	89	92	97	111	134	142
34		35	46	67	69	101	104	109	123	146	154
55		56	67	88	90	122	125	130	144	167	175
78		79	90	111	113	145	148	153	167	190	198
87		88	99	120	122	154	157	162	176	199	207
99		100	111	132	134	166	169	174	188	211	219
156		157	168	189	191	223	226	231	245	268	276

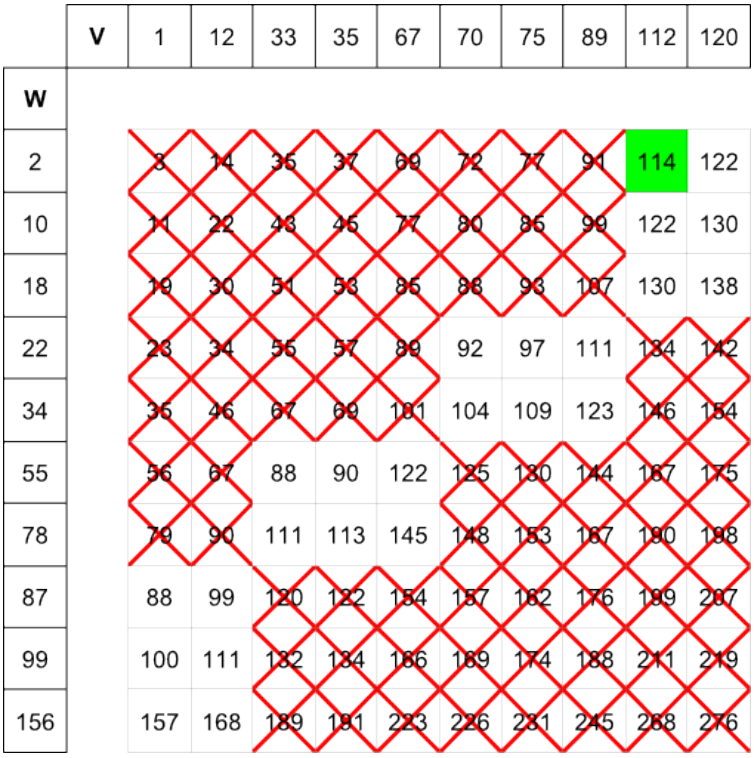
2) The result from the binary search will be 101.
Now we know that the inner square above 101 is lower than k and the bottom square is higher.
We can safely erase half of the input matrix here.

W \ V	1	12	33	35	67	70	75	89	112	120
2	3	14	35	37	69	72	77	91	114	122
10	11	22	43	45	77	80	85	99	122	130
18	19	30	51	53	85	88	93	107	130	138
22	23	34	55	57	89	92	97	111	134	142
34	35	46	67	69	101	104	109	123	146	154
55	56	67	88	90	122	125	130	144	167	175
78	79	90	111	113	145	148	153	167	190	198
87	88	99	120	122	154	157	162	176	199	207
99	100	111	132	134	166	169	174	188	211	219
156	157	168	189	191	223	226	231	245	268	276

3) Then we consider, recursively, the two inner square left as two new istances of the problem.
So we execute a binary search on both the diagonals.
Note: The algorithm is really easy to parallelize.

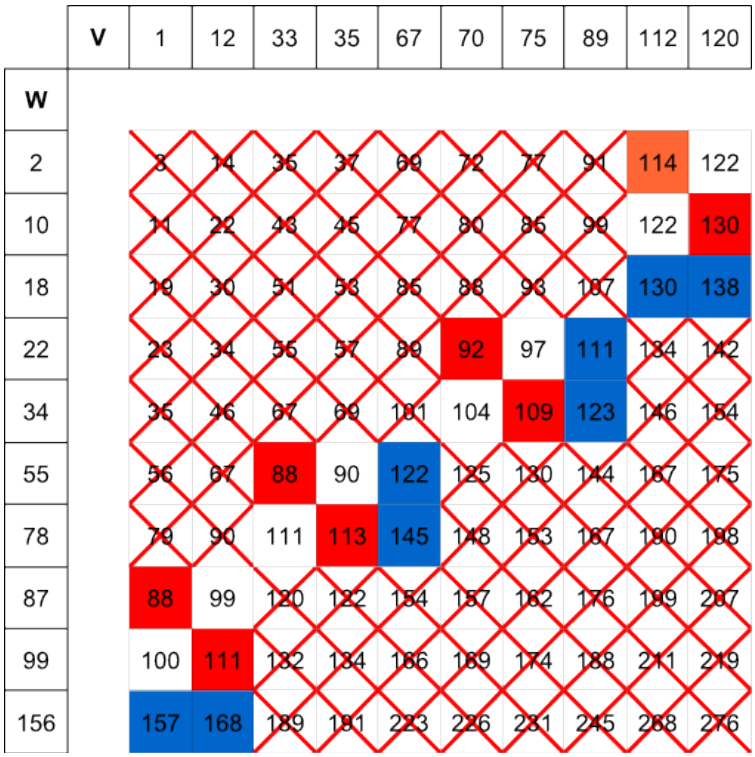
W \ V	1	12	33	35	67	70	75	89	112	120
2	3	14	35	37	69	72	77	91	114	122
10	11	22	43	45	77	80	85	99	122	130
18	19	30	51	53	85	88	93	107	130	138
22	23	34	55	57	89	92	97	111	134	142
34	35	46	67	69	101	104	109	123	146	154
55	56	67	88	90	122	125	130	144	167	175
78	79	90	111	113	145	148	153	167	190	198
87	88	99	120	122	154	157	162	176	199	207
99	100	111	132	134	166	169	174	188	211	219
156	157	168	189	191	223	226	231	245	268	276

4) Results will be 107 (above) and 90 (bottom) so we can safely erase more data and procede with the next step.



5) Finally we consider the four square left and for each of them we run a binary search on the diagonal. As shown, our k is in the first on top.

Note: Blue cells will be 8 new different istances of the problem.
The algorithm works only with square matrix, but when it finds a rectangular one, it simply splits it.



Pseudo code

```
diagonalize( matrix )
{
	if ( matrix is a single cell )
	{
		if ( k is in the cell )
			return k indexes;
		else
			return not found;
	}
	else if (matrix is rectangular)
	{
		split matrix;
		diagonalize ( square matrix );
		diagonalize ( what-is-left matrix);
	}
	else if (matrix is square )
	{
		temp = binarySearchOnDiagonal( k );
		if ( temp = k )
			return temp indexes;
		else if ( temp < k )
		{
			split matrix in half considering only the useful data;
			diagonalize( top-half matrix);
			diagonalize( bottom-half matrix);
		}
		else if ( temp > k )
		{
			split matrix in half considering only the useful data;
			diagonalize( top-half matrix);
			diagonalize( bottom-half matrix);
		}
	}
}
```

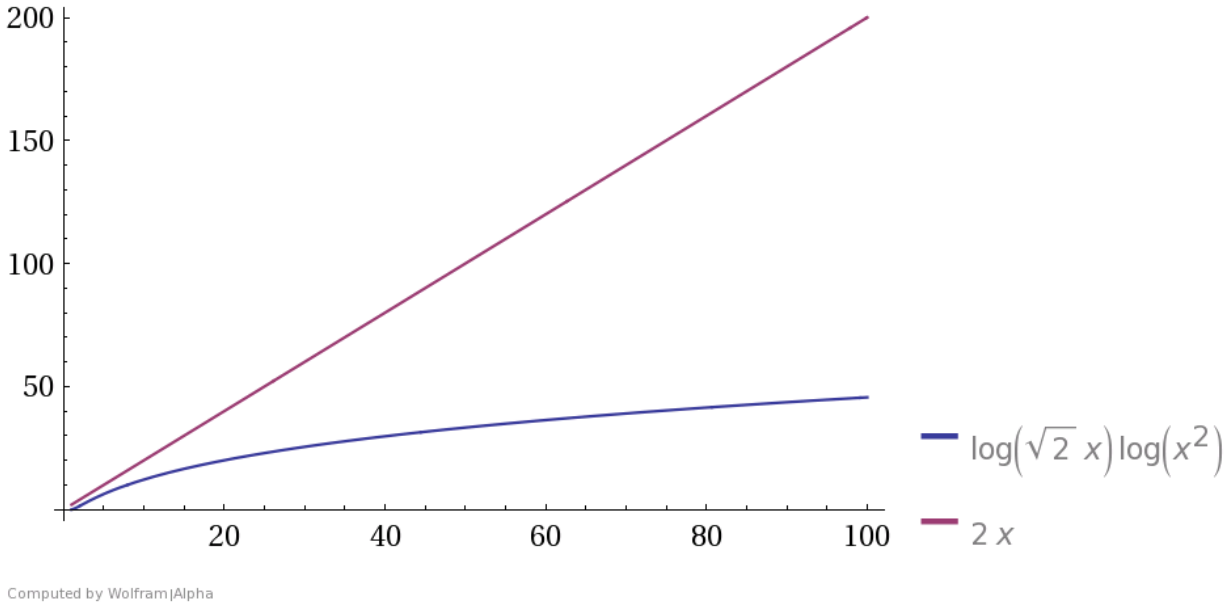
Computational cost

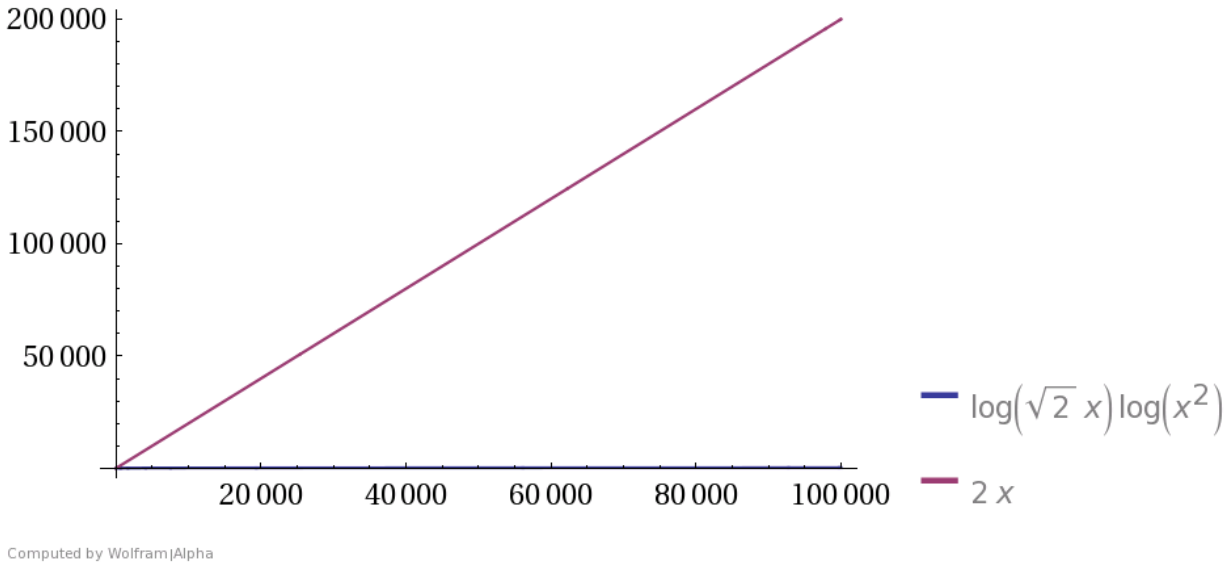
Let's consider n as V and W length for simplicity.

I'll consider the binary search cost on diagonals.
The algorithm is recursive.

In the first step, the matrix is $n \times n$ and has a $n * \sqrt{2}$ diagonal.
In the second step, the matrix is $\frac{(n \times n)}{2}$ and has two $\frac{n}{2} * \sqrt{2}$ diagonals.
In the third step, the matrix is $\frac{(n \times n)}{4}$ there are four $\frac{n}{4} * \sqrt{2}$ diagonals.

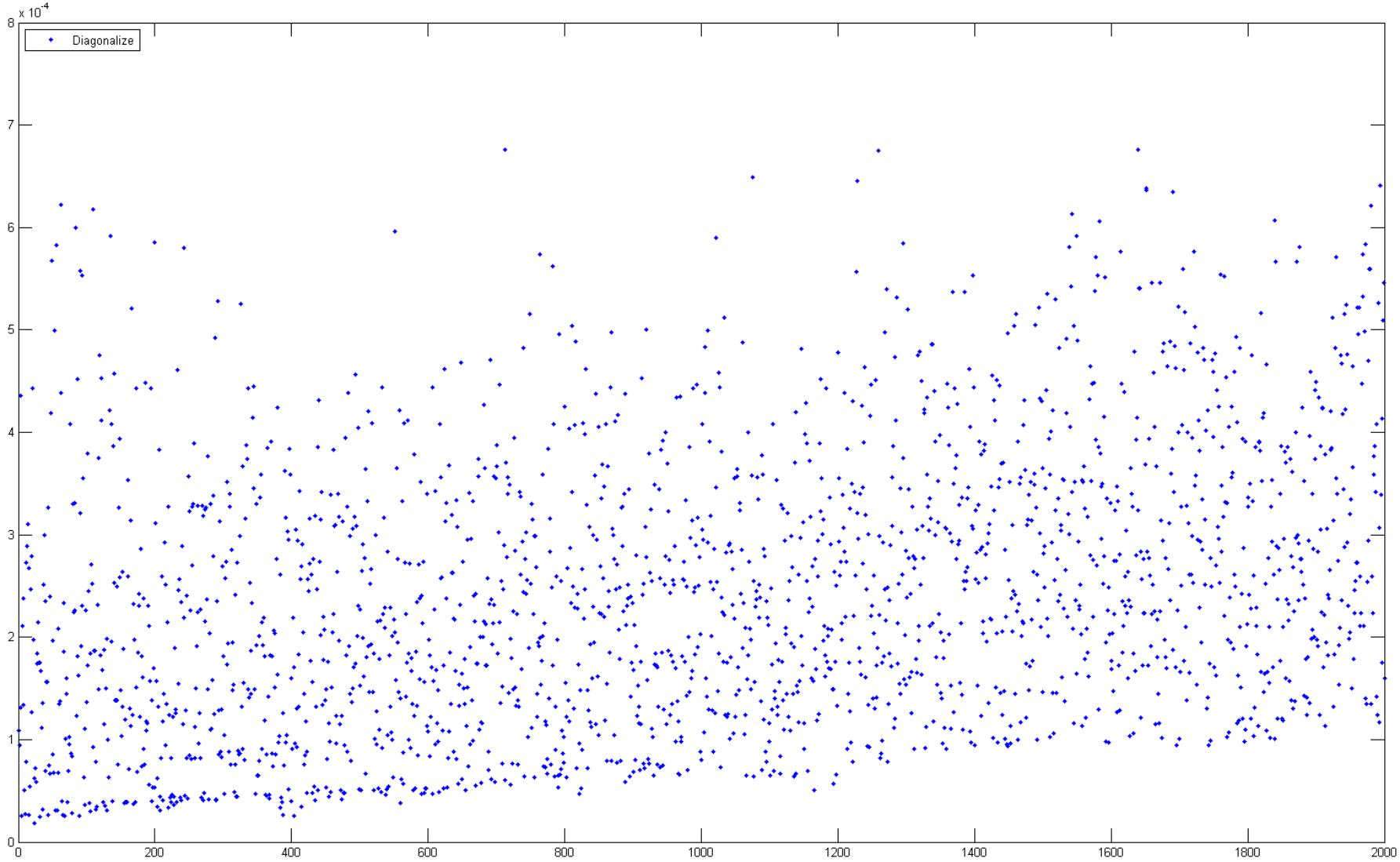
So, we'll need $\log(n^2)$ steps and in each of them, we'll run a binary search on the $\frac{i * n}{i} * \sqrt{2}$ diagonal.
The cost of the binary search is fixed and it's $\log(n * \sqrt{2})$.
The total computational cost of the algorithm is: $\log(n * \sqrt{2}) * \log(n^2)$

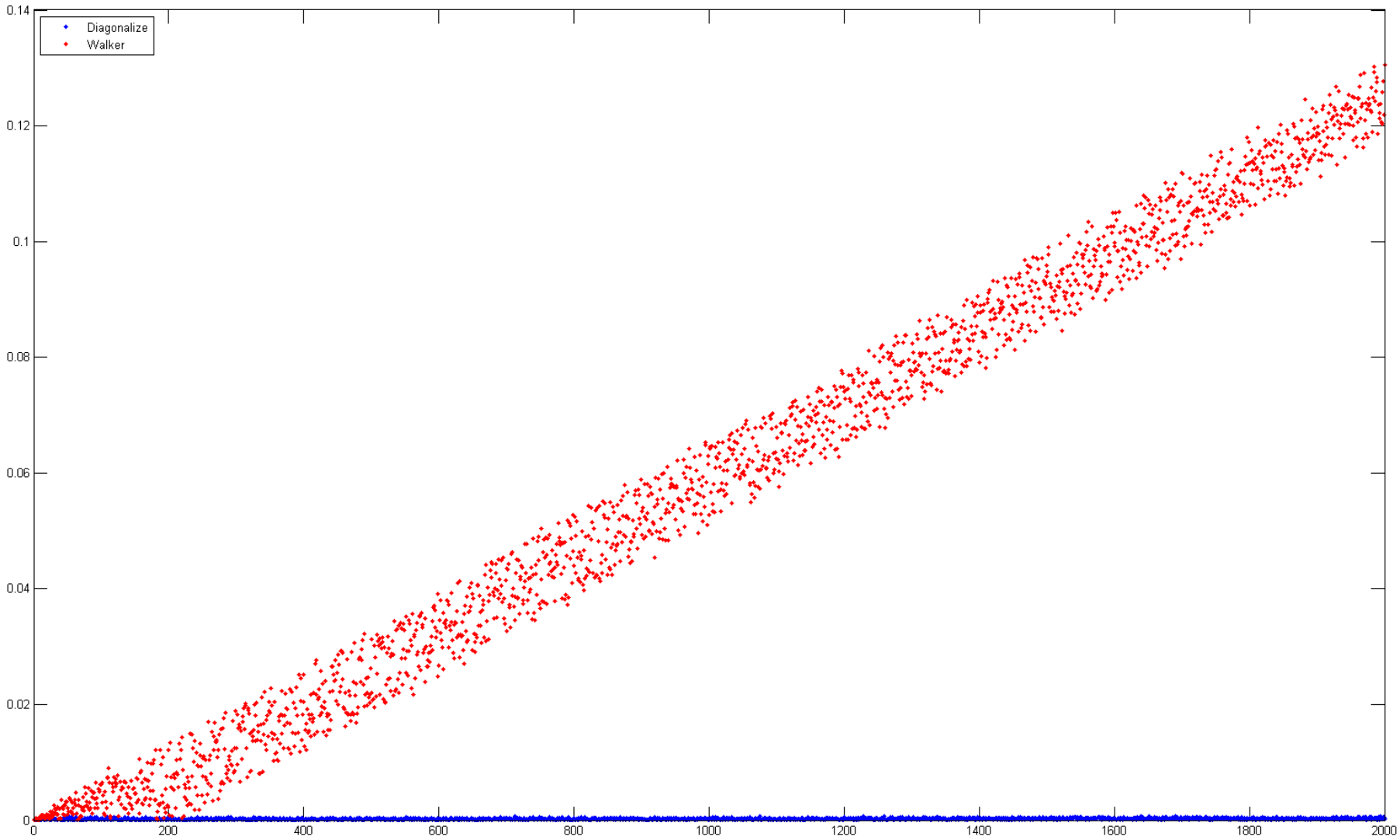




Final Test

To verify the correctness of the calculation of the computational cost, i used an experimental approach, comparing the times of Walker and Diagonalize.


x 10^-4
Diagonalize
8
7
6
5
4
3
2
1
0
0
200
400
600
800
1000
1200
1400
1600
1800
2000


Diagonalize
Walker
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
200
400
600
800
1000
1200
1400
1600
1800
2000