# Harmónikus oszcillátor

Márton Tamás

Eötvös Lóránd Tudományegyetem, Informatikus Fizikus

Számítógépes szimulációk laboratórium.

I. jegyzőkönyv.

# Tartalomjegyzék

# Ábrák jegyzéke

# 1. Fizikai probléma ismertetése

A fizikai problémák körében a harmonikus oszcillátor az a példa rendszer, amelynek tárgyalása majdnem az összes fizika előadáson előjön, hiszen ez azon fizikai problémakörök egyike, amelyet analitikusan könnyedén meg tudunk oldani. Gondoljunk csak a kvantummechanikára. A Schrödinger-egyenletet analitikus módon könnyen végigszámolhatjuk a hidrogénatom és a harmonikus oszcillátor esetében, de sajnos az élet nem csak ennyiből áll. Ezen motiváció alapján vezettük be a perturbációszámítás szükségességét a kvantummechanika kurzuson, hogy ne csak ezen két problémát tudjuk megoldani. Egy harmonikus oszcillátort ideális esetben, amilyenben a problémát mi is szimuláltuk és megoldottuk könnyedén elképzelhetjük. Gondoljunk egy rugóra egy asztalon, a súrlódástól eltekintve. Kitérítem a rugót, adok neki valamekkora kezdősebességet és meg is van a harmonikus oszcillátorunk. Matematikailag a harmonikus oszcillátort a potenciálja definiálja, ami a következő alakot ölti:

$$V(x) = \frac{1}{2} \cdot m \cdot \omega^2 \cdot x^2. \tag{1.1}$$

A mozgásegyenlete a harmonikus oszcillátornak:

$$m \cdot \ddot{x} = -m \cdot \omega^2 \cdot x. \tag{1.2}$$

Aminek analitikus megoldása a következő:

$$x(t) = x_0 \cdot cos(\omega \cdot t) + \frac{v_0}{\omega} \cdot sin(\omega \cdot t). \tag{1.3}$$

# 2. Megoldási módszerek

## 2.1. Az Euler-Cromer és Euler - eljárás

A fizikai probléma ismertetésénél láthattuk, hogy a harmonikus oszcillátor mozgásegyenlete egy másodrendű differenciálegyenlet, ami az $m$-el való egyszerűsítés után a következő alakot ölti:

$$\ddot{x} = -\omega^2 \cdot x, \tag{2.1}$$

mely átírható egy olyan alakra, ami kettébontható két elsőrendű csatolt differenciálegyenletre:

$$\frac{\partial^2 x}{\partial t^2} = -\omega^2 \cdot x \tag{2.2}$$

A csatolt differenciálegyenlet-rendszer:

$$\frac{\partial x}{\partial t} = v, \tag{2.3}$$

$$\frac{\partial v}{\partial t} = -\omega^2 \cdot x = a. \tag{2.4}$$

Első célunk az, hogy az analitikus megoldást összevessük a szimulációból numerikusan kiszámolt megoldással, ezért a csatolt, elsőrendű differenciálegyenlet-rendszert az Euler-Cromer-eljárással számoljuk.

Ami a következőt jelenti. Ha van egy kezdeti $x(t = 0)$ kitérésünk és egy $v(t = 0)$ kezdősebességünk amiket $dt$ időközökkel léptetünk, akkor az időfejlődésre a következő egyenleteket kapjuk:

$$v(t + dt) = v(t) + a(t)dt, \tag{2.5}$$

$$x(t + dt) = x(t) + v(t + dt)dt. \tag{2.6}$$

Az Euler-eljárással szemben az Euler-Cromer-algoritmusban az $x(t+dt)$ kifejezésben a sebesség a $t(t + dt)$ helyen értékelődik ki. Míg az Euler-eljárásban a mind ez t, helyen történik. Ami a programkódban úgí jelenik meg, hogy az alábbi sorrendben végezzük el:

$$x(t + dt) = x(t) + v(t + dt)dt. \tag{2.7}$$

$$v(t + dt) = v(t) + a(t)dt, \tag{2.8}$$

Ennek az az előnye, hogy az Euler-Cromer-eljárás esetén az energia megmarad és az alábbi alakot ölti:

$$E = \frac{1}{2} \cdot m \cdot v^2 + \frac{1}{2} \cdot m \cdot \omega^2 \cdot x^2 \tag{2.9}$$

megmarad.

Ezeket az eljárásokat a forráskód átírásával tudtuk elvégezni külön-külön.

```cpp
#include <cmath>
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
#include <bits/stdc++.h>

double omega;          // the natural frequency
double x, v;           // position and velocity at time t
int periods;           // number of periods to integrate
int stepsPerPeriod;    // number of time steps dt per period
string fileName;       // name of output file

void getInput();                // for user to input parameters
void EulerCromer(double dt);
void Euler(double dt);     // takes an Euler-Cromer step
double energy();                // computes the energy

void getInput ( ) {
    cout << "Enter omega: ";
    cin >> omega;
    cout << "Enter x(0) and v(0): ";
    cin >> x >> v;
    cout << "Enter number of periods: ";
    cin >> periods;
    cout << "Enter steps per period: ";
    cin >> stepsPerPeriod;
    cout << "Enter output file name: ";
    cin >> fileName;
}

void EulerCromer (double dt) {
    double a = - omega * omega * x;
    v += a * dt;
    x += v * dt;
}

void Euler (double dt) {
    double a = - omega * omega * x;
    x += v * dt;
    v += a * dt;

}

double energy ( ) {
    return 0.5 * (v * v + omega * omega * x * x);
}

int main ( ) {
    time_t start, end;
    time(&start);

    getInput();
    ofstream file(fileName.c_str());
    if (!file) {
        cerr << "Cannot open " << fileName << "\nExiting ...\n";
        return 1;
    }
    const double pi = 4 * atan(1.0);
    double T = 2 * pi / omega;
    double dt = T / stepsPerPeriod;
    double t = 0;
    file << t << '\t' << x << '\t' << v << '\t' << 0 << '\n';
    for (int p = 1; p <= periods; p++) {
        for (int s = 0; s < stepsPerPeriod; s++) {
            Euler(dt);
            t += dt;
            file << t << '\t' << x << '\t' << v << '\t' << energy() << '\n';
        }
        cout << "Period = " << p << "\tt = " << t
             << "\tx = " << x << "\tv = " << v
             << "\tenergy = " << energy() << endl;
    }
    file.close();
    time(&end);

    double time_taken = double(end - start);
    cout << "Time taken by program is : " << fixed << time_taken << setprecision(100);
    cout << " sec " << endl;
}
```

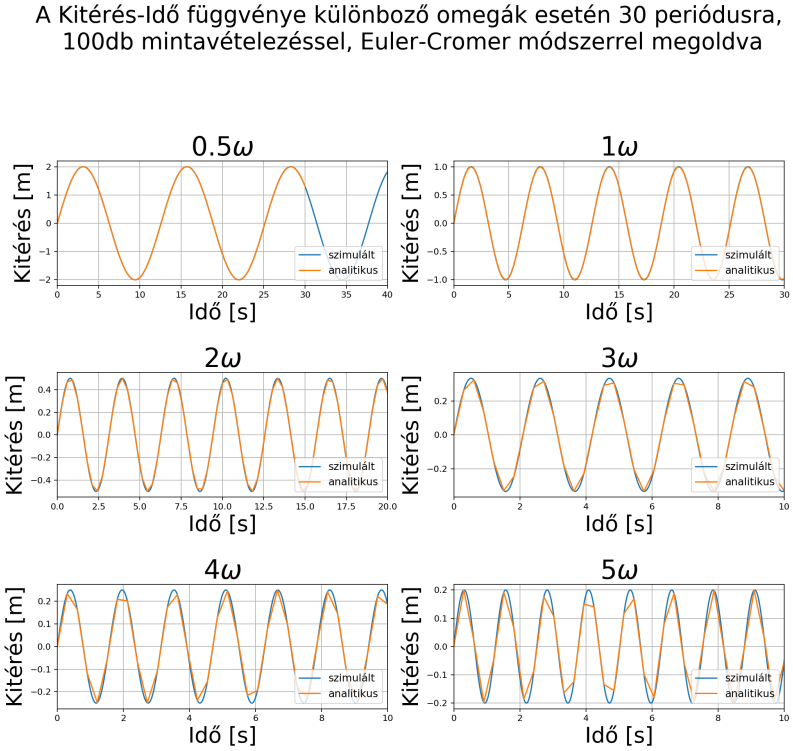
2.1.1. ábra. *A használt forráskód, mindkét módszerrel, valamint a futás idő méréséhez írt programrésszel*

# 3. Az eredmények értékelése

## 3.1. Kitérés-Idő diagramm vizsgálata

### 3.1.1. Euler-Cromer-algoritmussal számolva

A harmonikus oszcillátor szimulációjánál 30 periódust vizsgáltam, különböző lépésközökkel, különböző $\omega$-ákkal, valamint minden esetben $x_0 = 0$ és $v_0 = 1$ kezdeti feltételekkel.

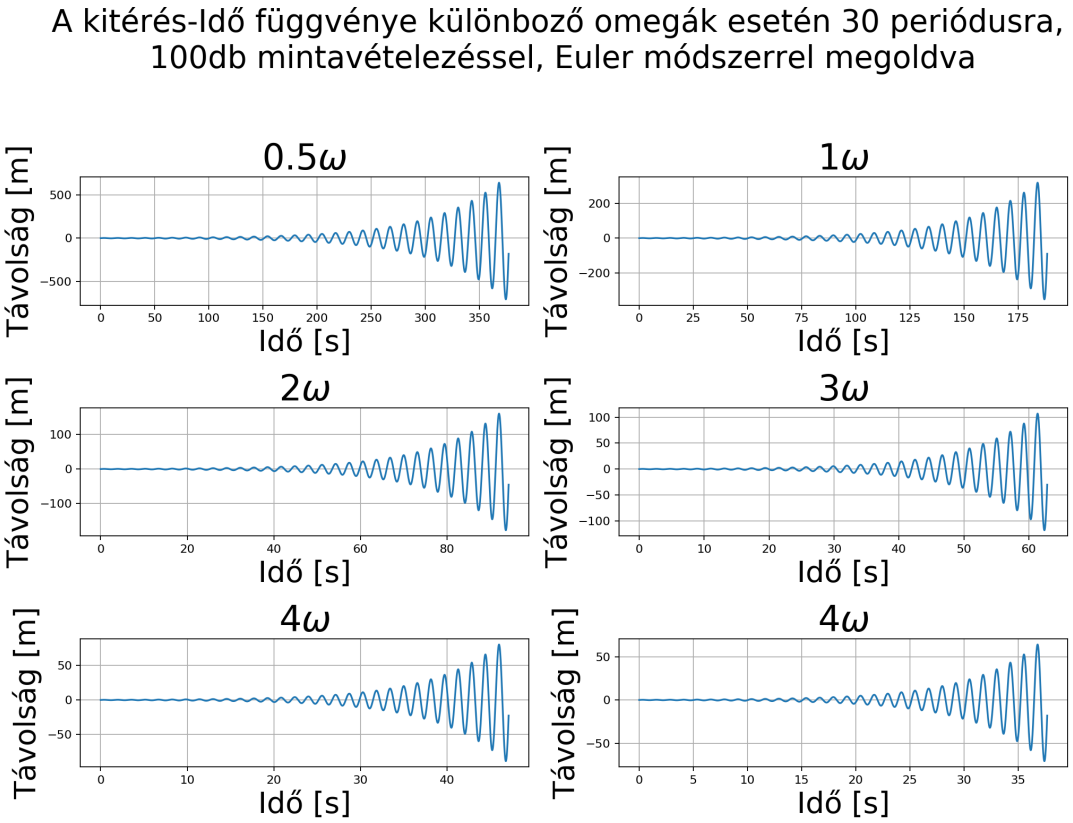



3.1.1. ábra. *A szimulált és analitikus megoldások kitérés-idő diagramjai*

Az ábrázolásokon jól megfigyelhető, ha összehasonlítva azonos $\omega$-ákhoz tartozó analitikus és Euler-Cromer-módszerrel számolt eredményeket, hogy változatlan lépésszámnál de magasabb $\omega$ értékekre, a két görbe nem egyezik. Amennyiben a mintavételezés számát emeljük arányosan, úgy ahogy $\omega$-át is, akkor úgy javítható az eredmény. Természetesen itt is a fő kérdés az, hogy melyik módszer éri meg jobban, melyik a költséghatékonyabb.

### 3.1.2. Euler-algoritmussal számolva

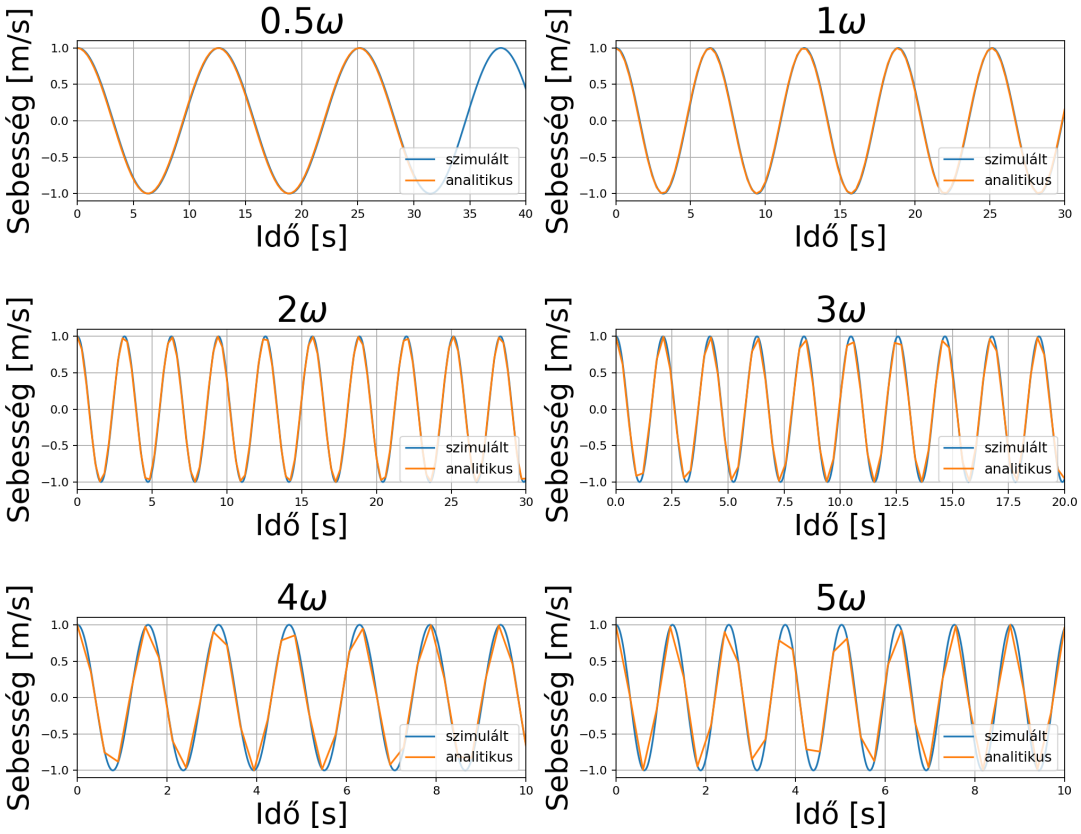
Euler-algoritmussal számolva jól látható, hogy a $t$ helyen történő kiértékelés következménye az, hogy a kitérés folyamatosan növekszik.




3.1.2. ábra. *A szimulált és analitikus megoldások kitérés-idő diagramjai*

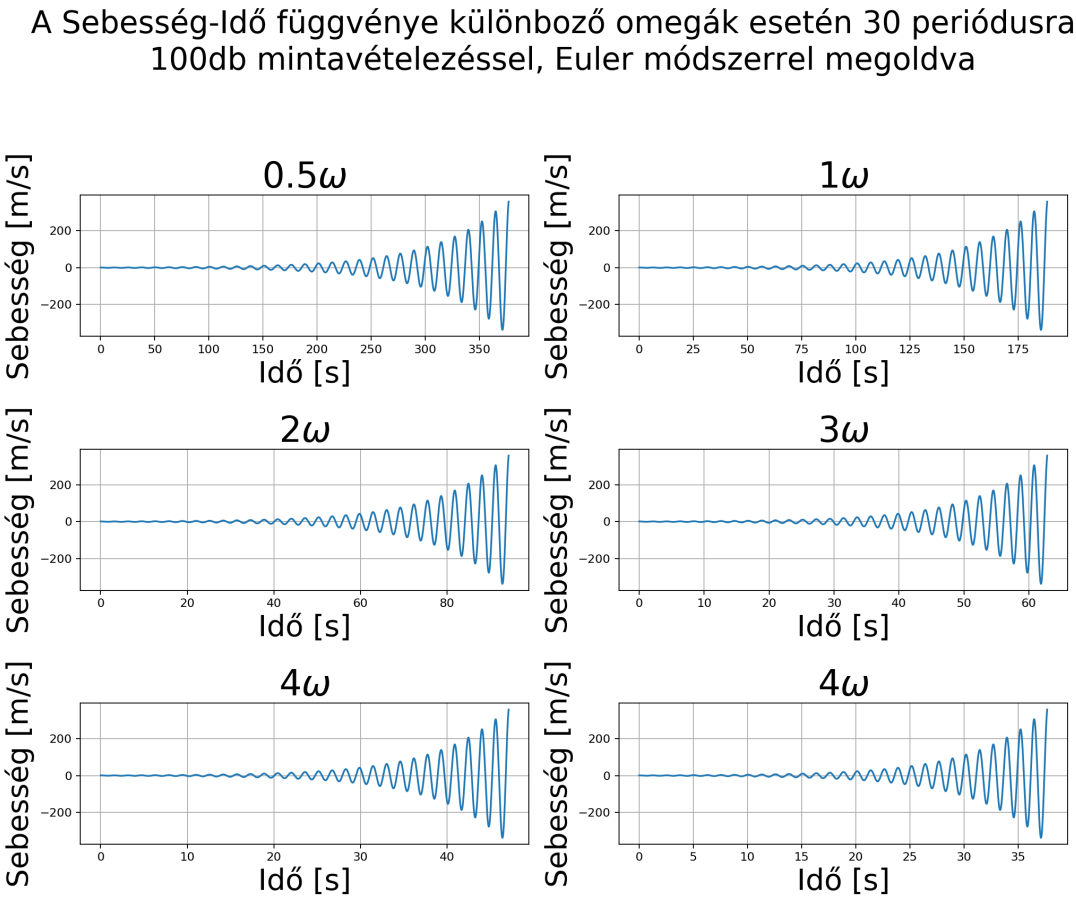
## 3.2. Sebesség-Idő diagramm vizsgálata

### 3.2.1. Euler-Cromer-algoritmussal számolva




3.2.1. ábra. *A szimulált és analitikus megoldások sebesség-idő diagramjai*

### 3.2.2. Euler-algoritmussal számolva

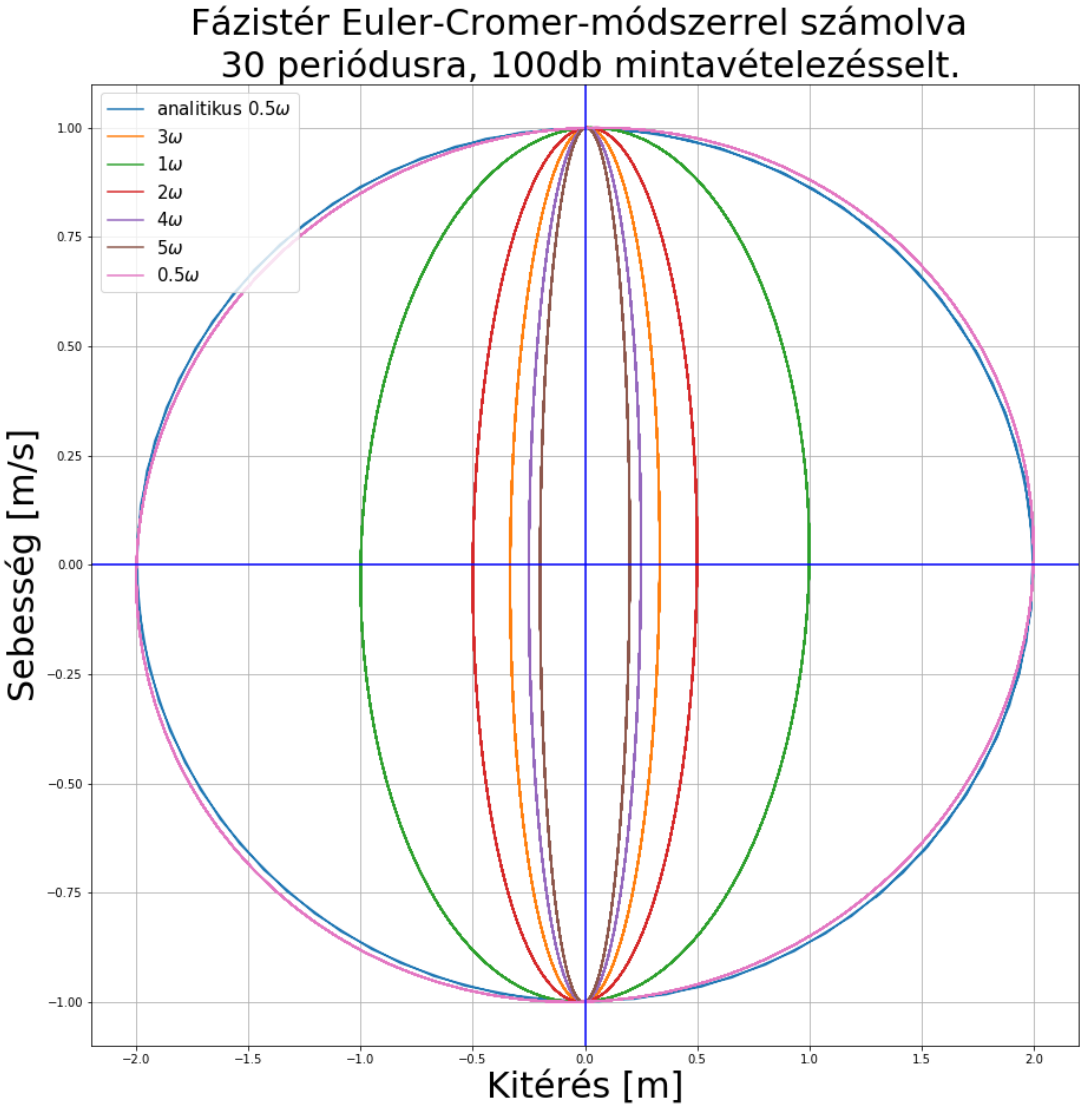



3.2.2. ábra. *A szimulált és analitikus megoldások sebesség-idő diagramjai*

## 3.3. Fázisgörbe vizsgálata

### 3.3.1. Euler-Cromer-algoritmussal számolva

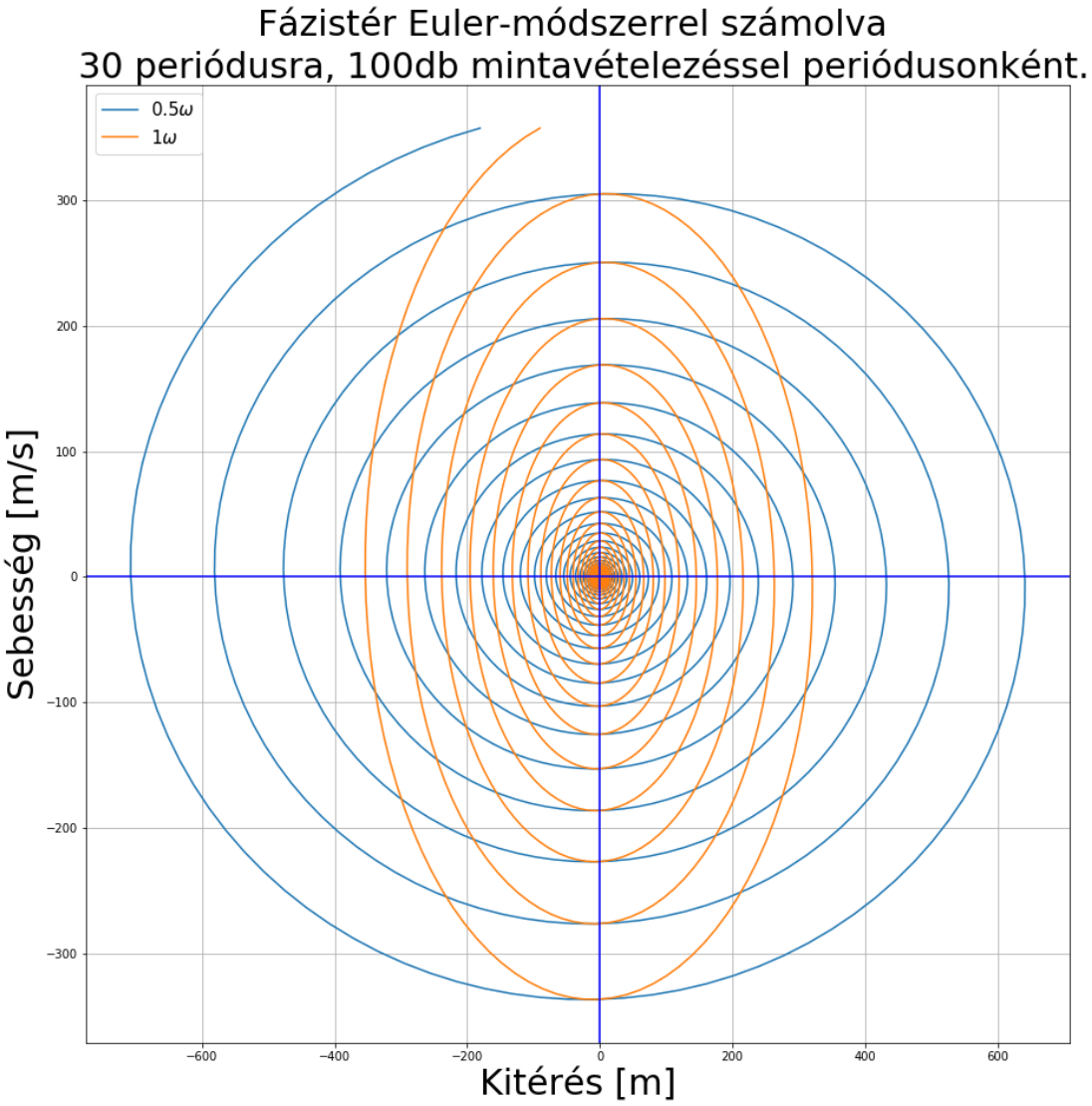
A fázisgörbe vitsgálatánál az $x_0$ és az $v_0$ kezdeti értékek ugyanazon $\omega$ és lépésszám mellet csak a görbe $x - y$ tengelyének nagyságát befolyásolják a periódus és a lépésszám hangolásával tudjuk elérni az analitikus megoldásnál kapott nem dőlt fázisgörbét. A lépésszám növelésével tudom a kiértékelésemet pontosítani, hogy az elméleti ellipszishez közelítsen. Ez jól látható az Euler-módszerrel számolt értékeknél.




3.3.1. ábra. *A szimulált és analitikus megoldások fázis diagramja*

### 3.3.2. Euler-algoritmussal számolva

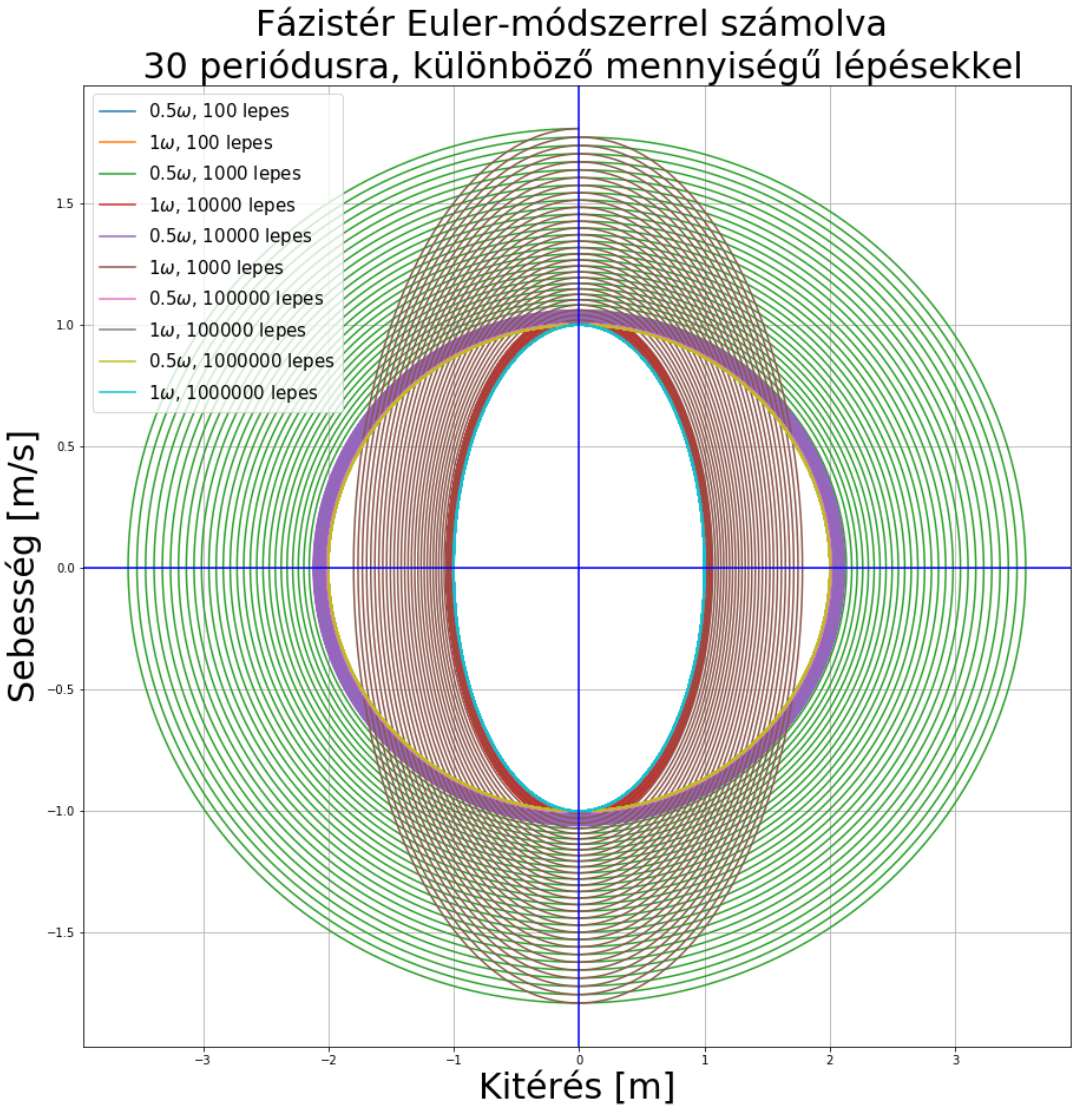
Euler-algoritmussal számolva, a a differenciál egyenletet megoldjuk és a lépésszám nem nagy akkor a $\Delta t$ nem kicsi, tehát nem elhanyagolható, így ezzel a $\Delta t$-vel mindig eltolódik a görbe.



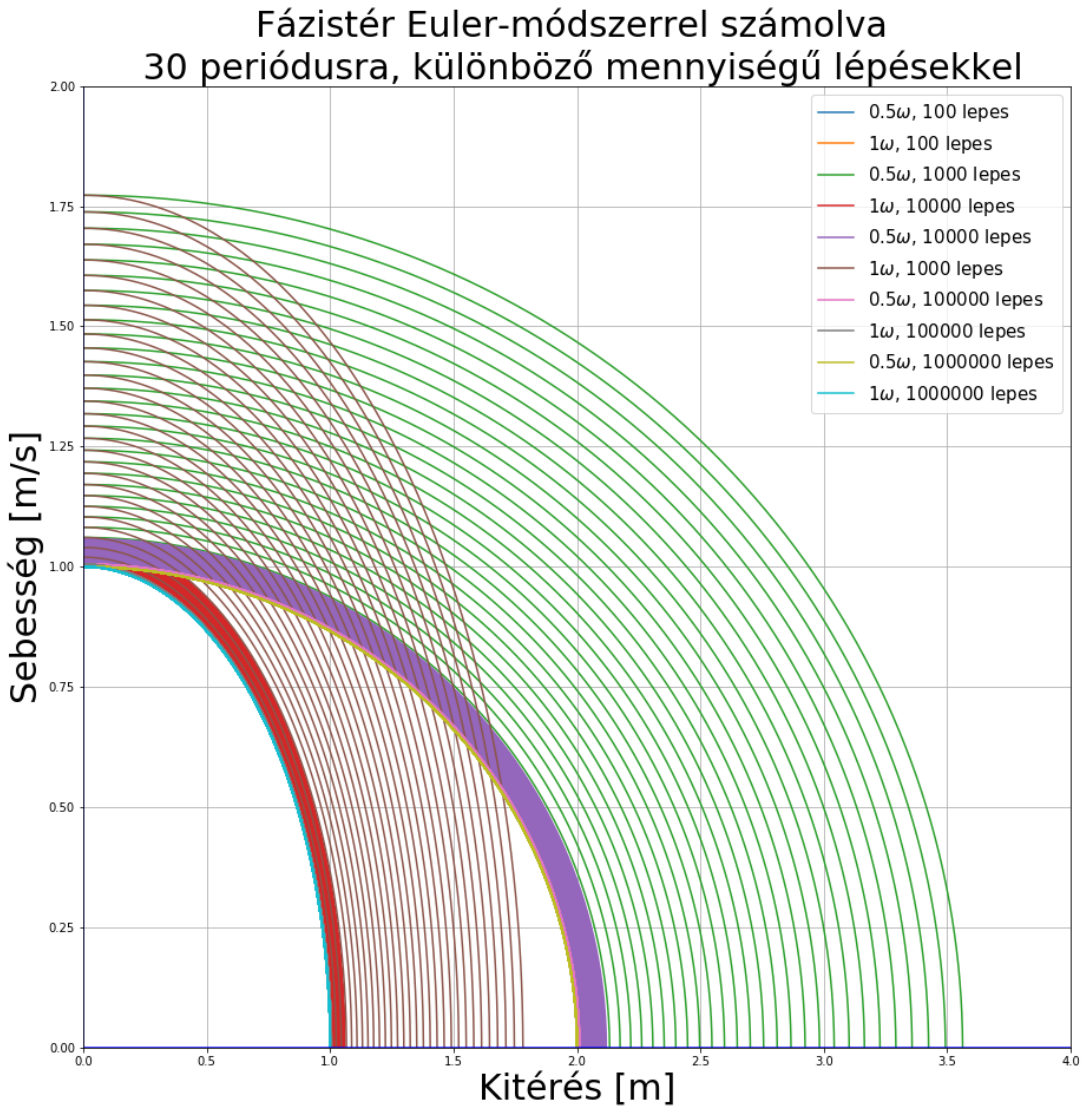

3.3.2. ábra. *A szimulált és analitikus megoldások fázis diagramja*

### 3.3.3. Euler-algoritmussal számolva a lépések növelésével

Amennyiben növelem a lépésszámot, úgy a $\Delta t$ mérete csökken, ami elhanyagolható nagyságúvá csökkenhet, így a két eljárás már egybevág.




3.3.3. ábra. *A szimulált megoldások fázis diagramja megnövelt lépésszámmal*



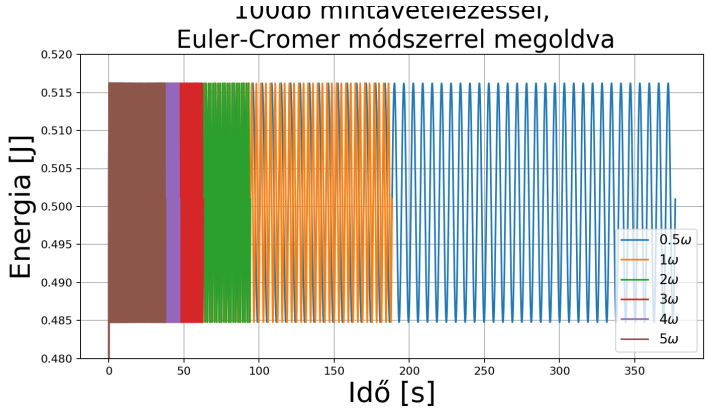

3.3.4. ábra. *A szimulált megoldások fázis diagramjának pozitív negyede megnövelt lépésszámmal*

## 3.4. Energia vizsgálata

### 3.4.1. Euler-Cromer-algoritmussal számolva

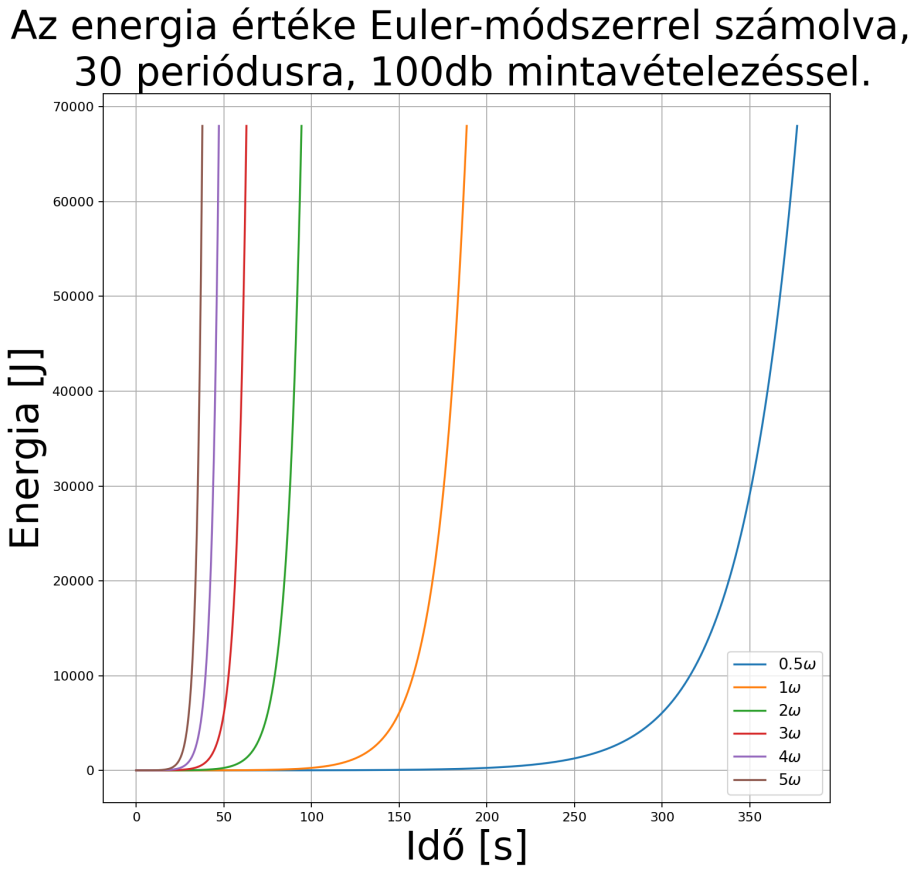
Tanulmányinkból tudjuk, hogy az energiának a hermonikus oszcillátornál meg kell maradni. Ez jól látszik az Euler-Cromer-eljárással számolt értékeknél már kis lépésszámnál is. Érdekesebb vizsgálat azonban, ha Euler-eljárással számolom az oszcillátor értékeit és úgy nézem az energiát. Jól látható, hogy azonos lépésszámoknál (100), az energia határozottam nem marad meg a végtelenbe tart. De ha ugyanezekkel az adatokkal ha 100000 lépéssel veszem a kiértékelést, akkor egy rövid felfutási idő

után az energia értéke a várt $0.5J$ értékre áll be minden esetben

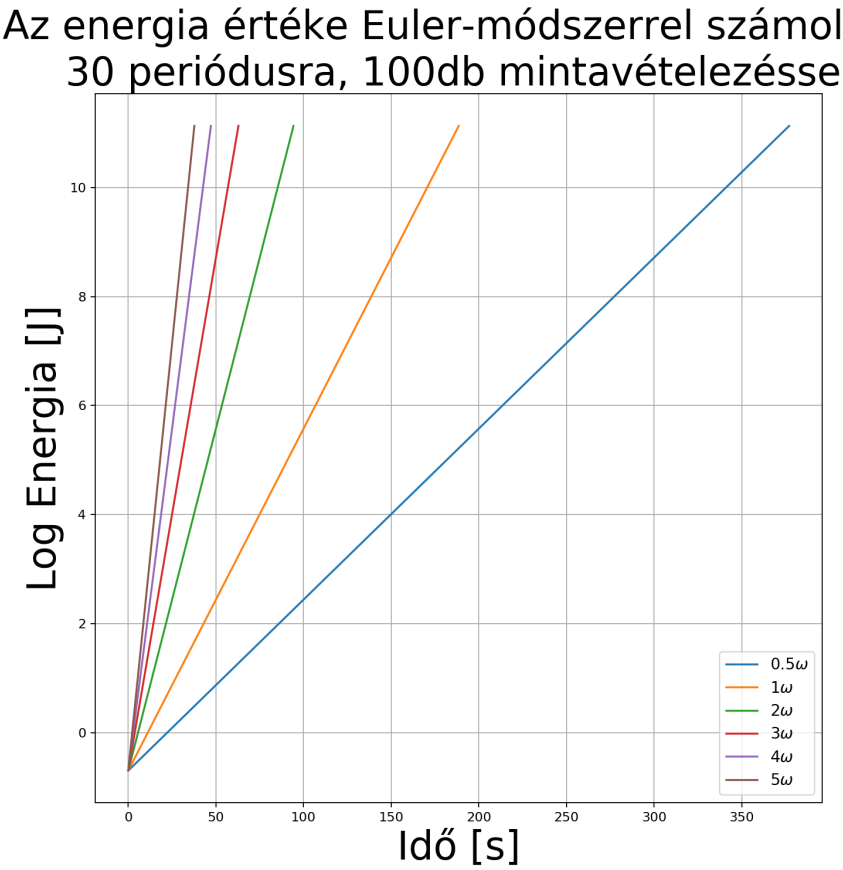



3.4.1. ábra. *A különböző ω-ákhoz tartozó energiák értékei.*

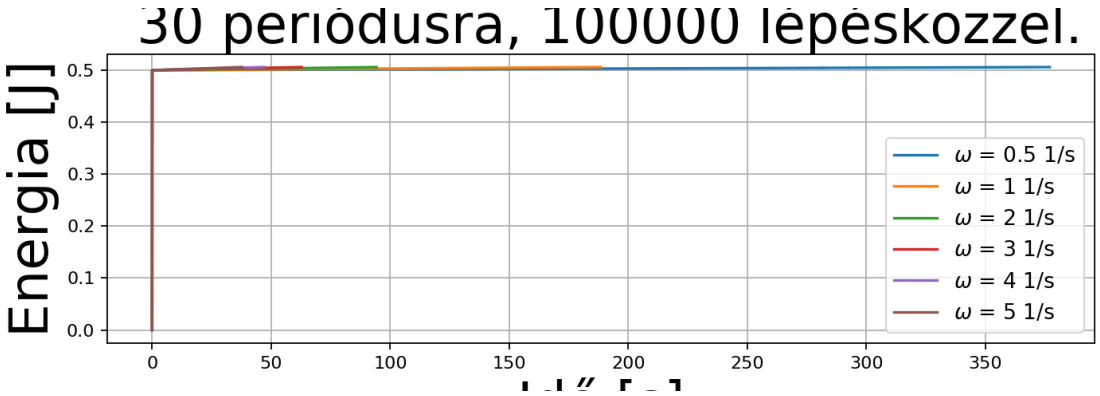
### 3.4.2. Euler-algoritmussal számolva




3.4.2. ábra. *A különböző ω-ákhoz tartozó energiák értékei.*

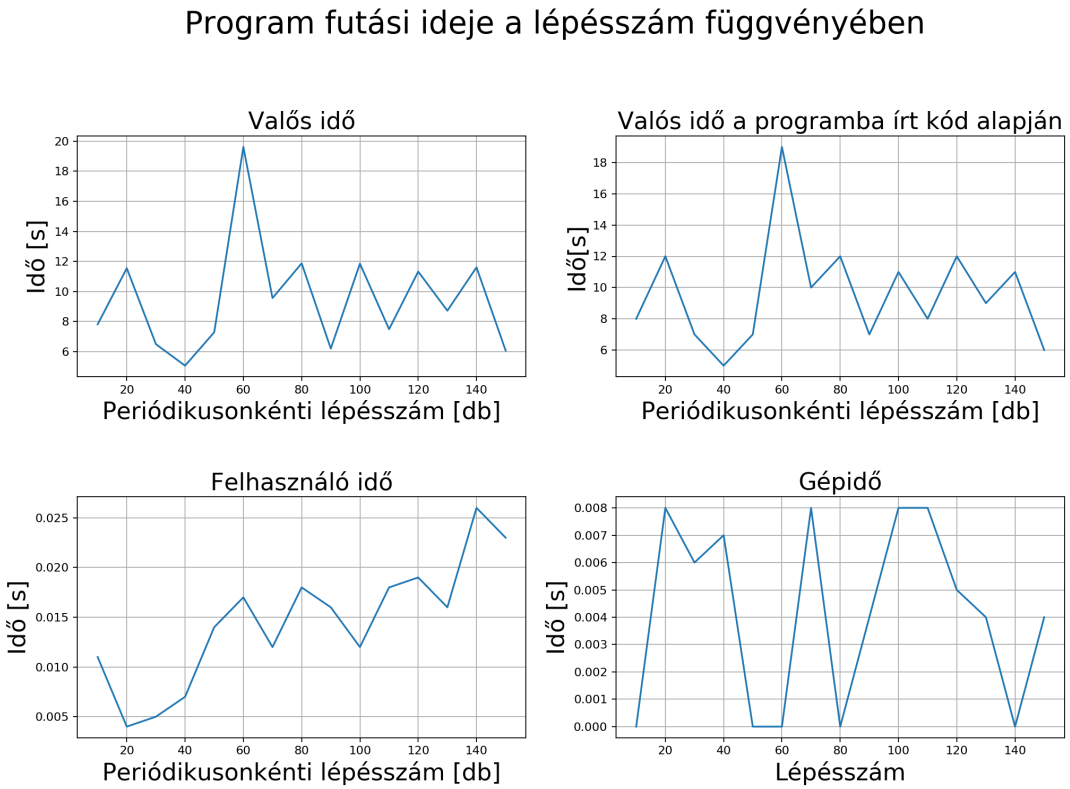



3.4.3. ábra. *A különböző ω-ákhoz tartozó energiák értékei logaritmikus skálán.*




3.4.4. ábra. *A különböző ω-ákhoz tartozó energiák értékei.*

### 3.5. Futás idő




3.5.1. ábra. *A futás idő periódusonkénti lépésszám függvényében.*

# 4. Diszkusszió

A jegyzőkönyvben bemutatom az Euler-Cromer, Euler és az Analitikus megoldások közötti különbséget, különböző lépésszámok és $\omega$-ák mellett. Amiből arra az eredményre jutottam, hogy a kívánt pontosság az összes módszerrel elérhető az $\omega$-ák és a lépésszám arányainak megfelelő beállításával. A kérdés csak az, hogy melyik a költség hatékonyabb számunkra, ami a megoldandó feladattól függ, hiszen a program futási ideje a lépésszámtól függ.