Számítógépes szimulációk
I.: 1D harmonikus oszcillátor

Pál Balázs[1]

[1]Eötvös Loránd Tudományegyetem

2019. február 12.

Abstract

A *Számítógépes szimulációk* című laboratórium első alkalmával az egyszerű, 1 dimenziós harmonikus oszcillátor mozgásegyenletének numerikus megoldását járjuk körül, mely a félévi tárgy bevezetőjeként szolgál. A szimuláció forráskódja C++ nyelven, az adatok elemzése és az ábrák pedig egy Jupyter Notebook-ban futó Python 3 kernel alatt készültek. A beadandó első célja a forráskóddal történő ismerkedés, valamint a paraméterek változtatásával történő különböző kezdőfeltételek beállítása és a szimuláció futtatása volt. Ezt követően az ebből kapott különböző kimenetek, megadott szempontok alapján történő elemzése volt a feladatunk.

1. BEVEZETÉS ÉS ELMÉLET

A Számítógépes szimulációk laboratórium első feladata során az egyszerű, 1-dimenziós harmonikus oszcillátor mozgását vizsgáltuk. Fizikai modelljének leírása és a szimuláció forráskódja már előzetesen a rendelkezésünkre állt a tárgy honlapján[1].
Az 1D harmonikus oszcillátort az alábbi mozgásegyenlettel írhatjuk le:

$$m \cdot \ddot{x}(t) = -m \cdot \omega^2 \cdot x(t) \tag{1}$$

Ennek a differenciálegyenletnek az analitikus megoldását egy megfelelő Ansatz segítségével kaphatjuk meg a legkönnyebben. Az egyenlet megoldása a következő lesz:

$$x(t) = x_0 \cdot \cos(\omega t) + \frac{v_0}{\omega} \cdot \sin(\omega t) \tag{2}$$

A numerikus megoldáshoz a mozgásegyenlet alábbi formáját használjuk fel:

$$\frac{d^2x(t)}{dt^2} = -\omega^2 \cdot x(t) \tag{3}$$

Ugyanis ezt felbonthatjuk a következő két, elsőrendű, csatolt egyenletre:

$$\frac{dx(t)}{dt} = v(t) \tag{4}$$

$$\frac{dv(t)}{dt} = -\omega^2 \cdot x(t) = a(t) \tag{5}$$

Ez az egyenletrendszer analitikusan is könnyen megoldható, azonban a gyakorlás kedvéért numerikus megoldáshoz folyamodunk.

2. FELADATOK

Az első szimuláció során négy darab feladatot volt szükséges teljesíteni a rendelkezésre álló forráskód fordítása után.
Az első a harmonikus oszcillátor kitérés-idő diagramjának felvétele tetszőlegesen hosszú időtartamra.
A második feladatban az oszcillátor kitérés-sebesség diagramját kell vizsgálnunk hosszú időtartamra. Itt elemeznünk és diszkutálnunk is kell a kapott eredményt, hogy miben tér el a várt képtől.
A harmadik feladat az Euler-Cromer és a szimpla Euler differenciálegyenlet megoldási módszerek összehasonlítása az energiamegmaradás szempontjából. A fő kérdés, hogy melyiknél és hogyan marad, vagy nem marad meg az energia?
A negyedik feladat során a szimuláció futásidejét kell tesztelnünk és megvizsgálnunk a megoldási módszerekben használt lépésszám függvényében.

3. MEGVALÓSÍTÁS

A szimuláció forráskódja C++ nyelven készült és az Euler-Cromer megoldási módszert alkalmazza, mely az Euler módszer apró bővítéssel ellátott verziója. Az Euler módszer lényege a következő, a fent ismertetett egyenleteken keresztül bemutatva[2]:

$$v(t + \Delta t) = v(t) - \omega^2 \cdot x(t) \cdot \Delta t \tag{6}$$

$$x(t + \Delta t) = x(t) + v(t) \cdot \Delta t \tag{7}$$

$$t \to t + \Delta t \tag{8}$$

Ennek módosított variációja a már említett Euler-Cromer módszer, ahol a második lépésben nem a t, hanem a $t + \Delta t$ helyen levő értékkel történik a kiértékelés. A módszer kontraintuitívnak tűnhet az alábbi módon:

$$v(t + \Delta t) = v(t) - \omega^2 \cdot x(t) \cdot \Delta t \tag{9}$$

$$x(t + \Delta t) = x(t) + v(t + \Delta t) \cdot \Delta t \tag{10}$$

$$t \to t + \Delta t \tag{11}$$

Mindkét esetben az $x(t = 0)$ és a $v(t = 0)$ értékekkel inicializáljuk a szimulációnkat, majd addig futtatjuk, míg a t érték el nem ér egy tetszőleges

t_{max} felső korlátot. Δt a szimuláció lépésközét jelöli, amit szintén mi választunk meg.
Várakozásaink szerint az utóbbi módszer előnye, hogy szinte megtartja az energiát, míg az első nem[3]. Ez persze egyéb módszerek segítségével még tovább javítható, azonban ennek a tárgyalásába most nem megyünk bele[4]. Az Euler és Euler-Cromer módszerek energiaviszonyainak vizsgálata majd a harmadik részfeladatban fog megtörténni.

3.1. A RENDELKEZÉSRE ÁLLÓ KÓD MODOSÍTÁSAI

Az eredeti forráskódon több változtatást kellett végrehajtani, hogy a kiszabott feladatokat meg lehessen vele oldani. Első lépésben az eredeti `sho.cpp` file-ból létrehoztam egy `sho_e.cpp` és egy `sho_ec.cpp` variációt. Az utóbbi az eredeti Euler-Cromer formulával megadott forráskód volt, míg az elsőben - az eredeti változtatásával - az Euler módszert implementáltam. A forráskódokat egy-egy segéd batch file segítségével fordítottam le, abban clang fordítót használva, mely kimeneti binárisa így egy `sho_e.exe` és egy `sho_ec.exe` lett. Az [1] linken elérhető forráskód ellenben egyéb pontokban is apró módosításokra szorult.
A második alapvető módosítás eredménye az, hogy a szimulációk végleges kimenetként egy *sho.dat* nevű file-t generálnak, ami az adatokat az 1. táblázatnak megfelelő elrendezésben tartalmazza.

t	$x\,(t)$	$v\,(t)$	$E\,(t)$	$T_{fut}\,(t)$
0	$x\,(0)$	$v\,(0)$	$E\,(0)$	$T_{fut}\,(0)$
Δt	$x\,(\Delta t)$	$v\,(\Delta t)$	$E\,(\Delta t)$	$T_{fut}\,(\Delta t)$
$2\Delta t$	$x\,(2\Delta t)$	$v\,(2\Delta t)$	$E\,(2\Delta t)$	$T_{fut}\,(2\Delta t)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_{max}	$x\,(\Delta t_{max})$	$v\,(\Delta t_{max})$	$E\,(\Delta t_{max})$	T_{fut}

1. táblázat. Az output `.dat` file-ok szerkezete

A következő változtatás motivációja a programok könnyebb futtathatóságának célja volt. Ezt megoldandó, a programok a szimulációs paramétereket terminálban megadott argumentumok formájában várják (ebből 5 darab van), amiket végül egy Juypter Notebook cella futtat az `os` package segítségével az alábbi módon, ahol az XX a tetszőlegesen választható `e`, vagy `ec` végződést helyettesíti:

```
os.system('sho_XX.exe ' + str(omega) + ' '
+ str(x_0) + ' ' + str(v_0) + ' ' + str(t)
            + ' ' + str(dt))
```

Ahol ω (omega) a rezgés körfrekvenciáját, x_0 (x_0) és v_0 (v_0) rendre a kezdő kitérés és sebesség nagyságát, t a szimulálandó periódusok számát, dt pedig a periódusonkénti lépésközt jelenti.
Az utolsó módosítás a negyedik feladatban szereplő időméréshez szükséges módosításokat foglalja magába. Ezt a C++ chrono library mikroszekundomos pontosságú időmérési módszerét felhasználva oldottam meg.
A véglegesített forráskód és az azt futtató Notebook elérhető GitHub-on[5].

4. KIÉRTÉKELÉS

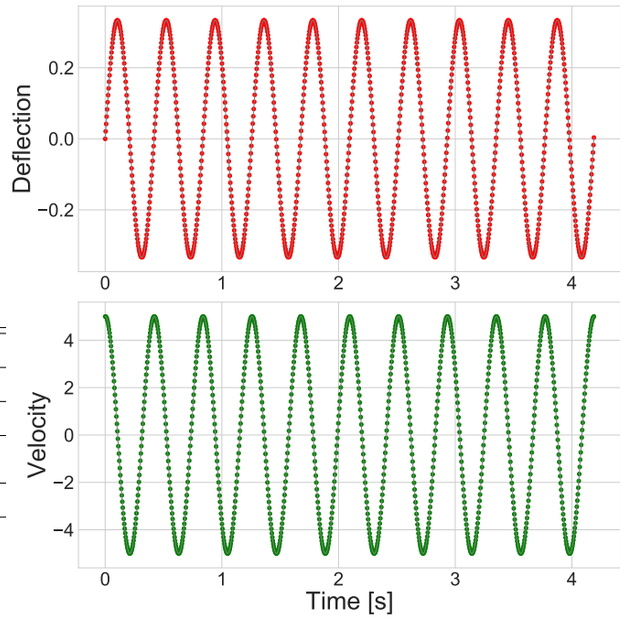
Az adatok elemzését egy Jupyter Notebook Python 3 kernelében végeztem el. Ez a notebook szintén elérhető a szimuláció GitHub repository-jában[5].

4.1. KITÉRÉS-IDŐ DIAGRAM

Az első feladatban a harmonikus oszcillátor kitérésének és sebességének viselkedését figyeltük az idő függvényében. A szimuláció kezdőfeltételei a 2. ábrán olvashatóak.

Szim. paraméter	Érték
ω	15
x_0	0
v_0	5
t	10
dt	100

2. táblázat. A szimuláció első feladatának kezdőfeltételei



1. ábra. Euler-Cromer módszer
Fent: Idő - Kitérés grafikon
Alul: Idő - Sebesség grafikon

A szimuláció során kapott adatok az első feladatnak megfelelően az 1. ábrán láthatóak. Ez a mélyebb megértés céljából egy sebesség-idő grafikonnal is ki lett bővítve. A kapott kép alapján az elméletből elvártakat kaptuk vissza. Az oszcillátor sebessége zéró kitérésnél a legnagyobb, maximális kitérésnél 0, majd pedig előjelet vált.

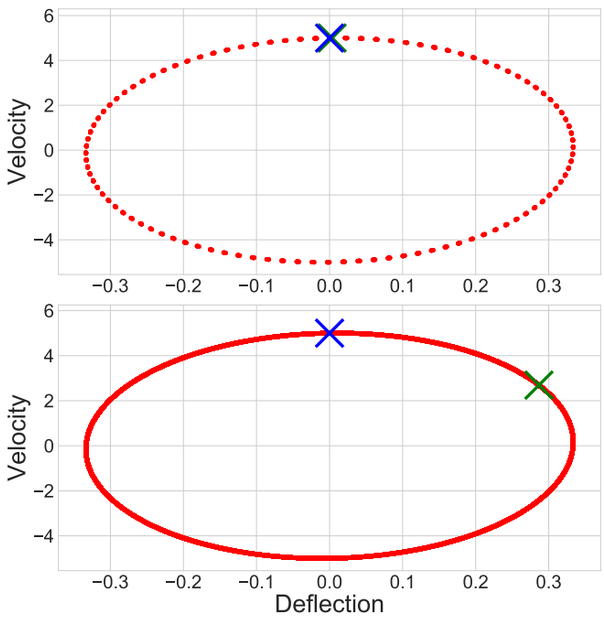
4.2. KITÉRÉS-SEBESSÉG DIAGRAM

A második feladatban a szimulációt nagyon hosszú idejű futtatásra kellett vizsgálni. Az elméleti ismertetőben már szó volt róla, hogy az Euler-Cromer módszer a natív Euler módszernél jobban, teljesen megőrzi az energiát. Az elmélet szerint azt várnánk, hogy bármilyen hosszú idejű futtatásra is, a kitérés-sebesség diagram egy ellipszist adna eredményül, mely ugyanabba a pontba tér vissza egész számú periódusra vizsgálva. Mivel azonban a differenciálegyenlet ezen megoldási módszere sem teljesen tökéletes, így ez hosszabb idejű futtatásra már

nem lesz igaz. Míg rövid idejű futás esetén ($\sim$ 10 periódus) nem lesz számottevő eltérés, addig nagyobb számú periódus ($\sim$ 1000) esetén a kitérés-idő diagram nem ugyanabba a pontba fog visszatérni. A két eset összehasnolítása a 2. ábrán látható, a hozzá felhasznált szimulációs paraméterek pedig a 3. táblázatban találhatóak.

Szim. paraméter	Rövid futás	Hosszú futás
ω	15	15
x_0	0	0
v_0	5	5
t	10	1000
dt	100	100

3. táblázat. A szimuláció második feladatának kezdőfeltételei

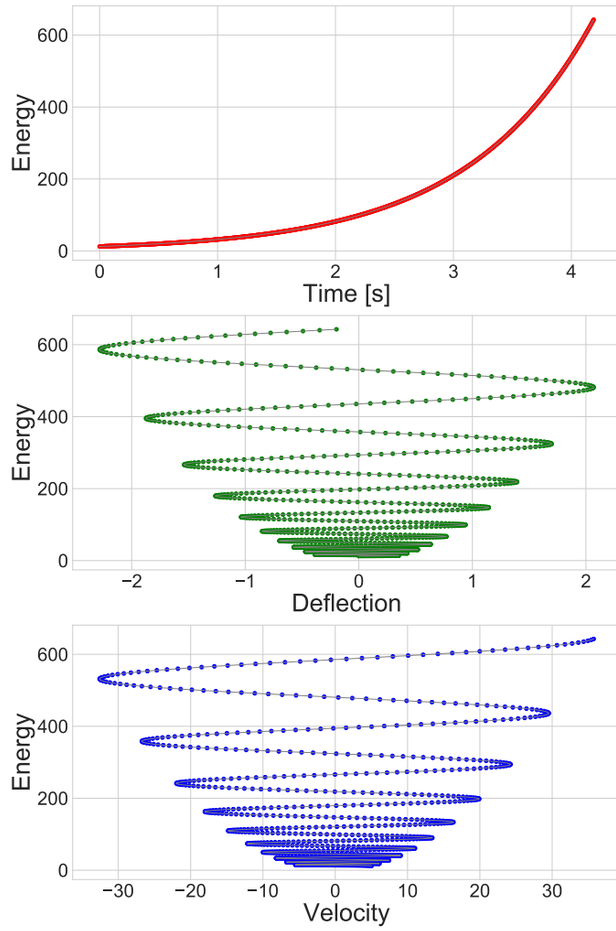


2. ábra. Euler-Cromer rövid és hosszú futás
Fent: Kitérés - Sebesség grafikon rövid futásidő esetén
Alul: Kitérés - Sebesség grafikon hosszú futásidő esetén

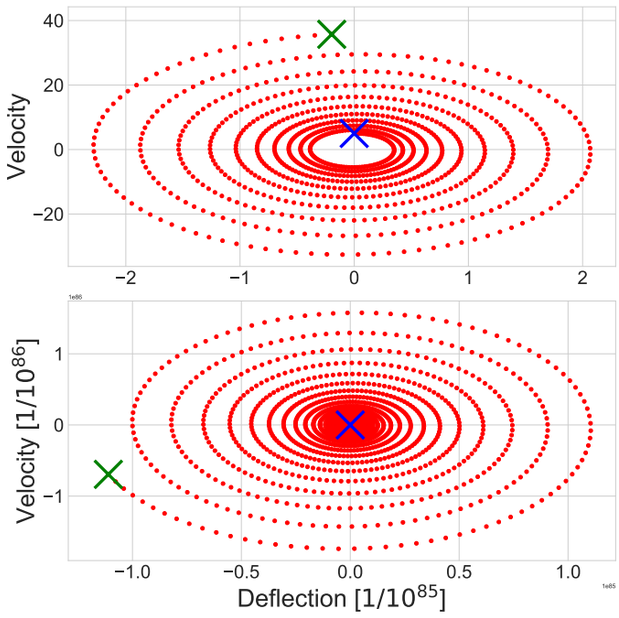
A 2. ábrán a kék X a szimuláció kezdőpontját míg a zöld X a végpontját jelenti. Míg az első ábrán teljesen fedik egymást, addig a másodikon tisztán látszik, hogy a periódusok nagy mértékben „elcsúsznak”.

4.3. ENERGIAMEGMARADÁS

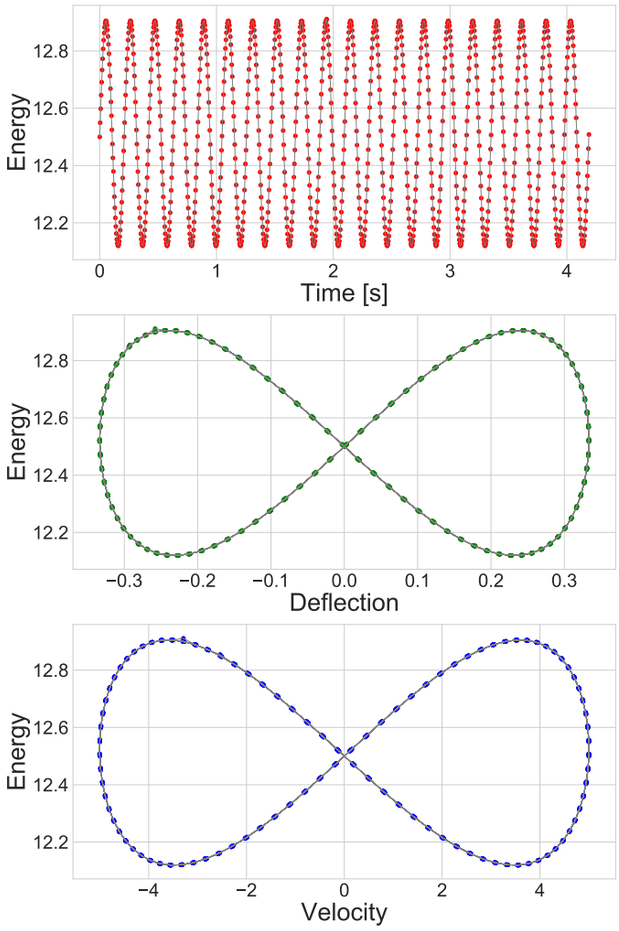
A harmadik feladatban a natív Euler és az Euler-Cromer differenciálegyenlet megoldási módszereket kellett összehasonlítanunk az energiamegmaradás szempontjából. Ez szorosan kapcsolódik az előbbi feladathoz, ugyanis itt is szintén a szimuláció hosszabb távú pontossága a kérdés.
Hogy összehasonlíthassuk a két módszert, az eredeti - már módosított - file-ról másolatot készítettem és az Euler-Cromer módszer módosításával abban implementáltam a natív Euler módszert. Ezt a két programot szimultán futtattam, azonos, az 1. táblázatban szereplőekkel megegyező paraméterekkel, amiket újfent egy Jupyter Notebook-ban futó Python 3 kernel segítségével hasonlítottam össze.
Az Euler módszer esetén az energia nem marad meg, folyamatosan növekszik az idő előrehaladtával. Ez abból adódik, hogy az oszcillátor kitérése és sebessége a nem teljesen optimális léptetés miatt szintén folyamatosan növekszik, az energia pedig ismert módon ezek által határozódik meg. A 3. ábrán az így futtatott program idő-energia, kitérés-energia és sebesség-energia grafikonjait láthatjuk. A szimuláció kezdőfeltételei megegyeznek az 2. Az 4. ábrán érdekességképpen az Euler módszerrel kapott kitérés-sebesség grafikont ábrázoljuk. A 2. ábrához hasonló jelölés segítségével látható, hogy a várt ellipszis helyett egy kifelé terjedő spirált kapunk. Ezzel összehasonlítandó, az Euler-Cromer módszer segítségével lefuttatott szimuláció energiaviszonyairól a 5. ábráról tájékozódhatunk.



3. ábra. Euler módszer
Fent: Idő - Energia
Középen: Kitérés - Energia
Alul: Sebesség - Energia



4. ábra. Euler rövid és hosszú futás
Fent: Kitérés - Sebesség grafikon rövid futásidő esetén
Alul: Kitérés - Sebesség grafikon hosszú futásidő esetén



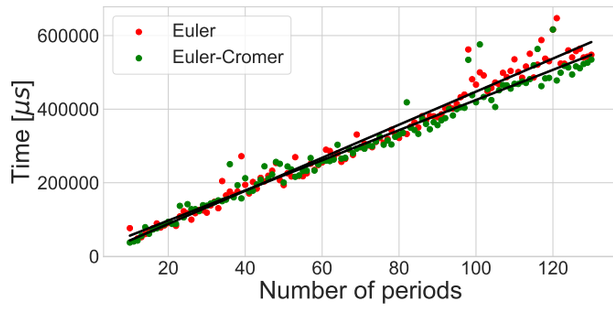
5. ábra. Euler-Cromer módszer
Fent: Idő - Energia
Középen: Kitérés - Energia
Alul: Sebesség - Energia

4.4. FUTÁSIDŐ

A negyedik és egyben utolsó feladat a program futásidejének vizsgálata volt a szimuláció lépésszámának függvényében. A lépésszámot két módon növelhetjük:

1. A periódusok számának növelésével
2. A periódusonkénti dt lépésköz hosszának csökkentésével

Jelen esetben az egyszerűség kedvéért az első módszer segítségével határoztam meg a futásidő változását. Az egyéb paramétereket a 3. táblázatban szereplőkkel megegyezően vettem fel. Az idő méréséhez a chrono library `std::chrono::steady_clock` `std::chrono::microseconds`, mikroszekundum pontosságú függvényét használtam. A szimulációs paraméterek itt is az 1. táblázatban szereplőekkel azonosak voltak, azonban a futásidőket 10 és 130 darabszámú periódus közti futtatásokra vizsgáltam. A kapott értékek a 6. ábrán láthatóak.



6. ábra. Futásidő
Fent: Euler módszer
Alul: Euler-Cromer módszer

Az Euler-Cromer módszer esetében az adatok alapján láthatóan valamivel jobb futásidő érhető el hosszabb távon, azonban ezt a különbséget befolyásolhatták a szimulációt végző számítógépen futó egyéb folyamatok is.
Végérvényben biztosra csak azt mondhatjuk, hogy mindkét módszer esetén a periódusok számának növelésével a futásidő lineárisan növekszik. Az Euler módszer esetén a futásidőt az alábbi függvény adja meg a mérések során:

$$T(P) = (4493 \cdot P - 1736)\ \mu s \tag{12}$$

Míg az Euler-Cromer módszer esetében:

$$T(P) = (4097 \cdot P + 15184)\ \mu s \tag{13}$$

Ahol P a periódusok darabszámát jelöli.

5. DISZKUSSZIÓ

A Számítógépes szimulációk laboratórium első gyakorlatán kitűzött témához - az 1D harmonikus oszcillátorhoz - tartozó feladatokat kivétel nélkül sikerült a rendelkezésre álló időn belül megoldani. A vele lévő munka során felmerülő akadályokat - mind a LaTeX, mind a `C++` és `Jupyter/Python` programozás terén sikerült áthidalni.
Az első feladatban kizárólag az Euler-Cromer módszerhez tartozó kitérés-idő és sebesség-idő grafikont csatoltuk, azonban a gyakorlathoz létrehozott GitHub repository-ban[5] található notebookban az Euler módszerhez tartozó is megtalálható. A futásidők vizsgálatánál a két módszer közötti eltérést biztosra nem tudom megmondani, hogy mi okozhatja. Mivel egy elég gyenge laptopon futott azok kiértékelése, ezért lehetséges, hogy valamilyen épp lefutó háttérprogram zavarta meg a kapott adathalmazt. Tisztán látszott azonban, hogy a mindkért esetben lineáris a függés a szimulált periódusok száma és a futásidő között.

6. MEGJEGYZÉSEK

A dokumentumban található képek 150 dpi felbontású, `.pdf` kiterjesztésű forrásból származnak, amiket a `matplotlib` package `savefig` függvényének segítségével mentettem le. Ezek miatt a jegyzőkönyv néha lassan tölt be, azonban a már többször említett GitHub repository-ban az nagyon könnyen olvasható. (Ennek az oka feltehetően az, hogy GitHub-on megnyitva az oldal az egész PDF-et betölti, míg egy offline PDF olvasó csak egy adott tartományban levő részeket az optimalizáció miatt.) A további jegyzőkönyvek lehetőleg szintén ezt a formátumot fogják követni.
Remélhetőleg az itt elsajátított tudást sikeresen alkalmazhatom majd a további problémák megoldásához is.

FORRÁSOK

[1] Stéger József. *Számítógépes szimulációk gyakorlat [szamszimf17la]*. [Online; opened at 2019.02.12.] 2019. URL: https://stegerjozsef.web.elte.hu/teaching/szamszim/index.php.

[2] Csabai István. *PHY 410-505 Computational Physics I — Chapter 3: Oscillatory Motion and Chaos*. [Online; opened at 2019.02.12.] 2008. URL: http://csabai.web.elte.hu/http/szamszim/ch3-lec1.pdf.

[3] Alan Cromer. "Stable solutions using the Euler approximation". In: *American Journal of Physics* 49.5 (1981), pp. 455–459. DOI: 10.1119/1.12478. eprint: https://doi.org/10.1119/1.12478. URL: https://doi.org/10.1119/1.12478.

[4] Ernst Hairer, Christian Lubich, Gerhard Wanner. "Geometric numerical integration illustrated by the Störmer–Verlet method". In: Acta Numerica. p. 399–450. Cambridge University Press, 2003. ISBN: 9780511550157. DOI: 10.1017/cbo9780511550157.006.

[5] Pál Balázs. *ELTE Computer Simulations 2019 — GitHub*. [Online; opened at 2019.02.13.] 2019. URL: https://github.com/masterdesky/ELTE_Comp_Simulations_2019.