Ingák szimulációja közegellenállással

Tartalomjegyzék

1. Bevezetés, használhatóság

A projekt során először egy inga mozgásának, majd pedig két darab inga együttes viselkedésének leírására és szimulálására törekedtem. A hiteles fizikai kép miatt a közegellenállást figyelembevéve kellett az inga időben változó paramétereit kiszámítani/szimulálni, és ezeket az értékeket egy szabványméretű konzolon kellett ábrázolni. Továbbá, ezek mellett a program képes két inga együttes viselkedésének szimulációja során, reális módon kezelve, az ingák ütközését pár százalékos, relatívan alacsony hibán belül leírni és szimulálni. A hiba csökkentése orvosolható a hibahatárok csökkentésével a programban, azonban ez nagyon megnöveli a futásidőt, emiatt a beállított határokon belül érdemes a programot futtatni. A felhasznált fizikai törvények pontosabb kifejtéséről a későbbiekben olvashatunk.

2. Szükséges környezet, könyvtárak

Az alapvető működéshez elengedhetetlen egy operációs rendszer megléte, ajánlott főként az ismeretebbek közül (Linux, Windows, MacOS) valamelyik feltelepítése. A program elsősorban Windows 11-en lett tesztelve, azonban működni kell további operációs rendszereken is, amennyiben azon létezik valamely C++ nyelv fordításához szükséges eszköz. A program alapvetően a Code::Blocks nevezetű fejlesztői környezetben készült el C++ nyelven. Maga a program nem igényel semmifajta külső könyvtárat, csak a C++, C nyelvbe alapvetően beépített könyvtárak (iostream, cstdio, math.h, stdexcept, fstream, string) kerülnek felhasználásra, valamint alapvető C++ nyelvi elemek.

3. A projekt alapötlete és a megvalósítás menete

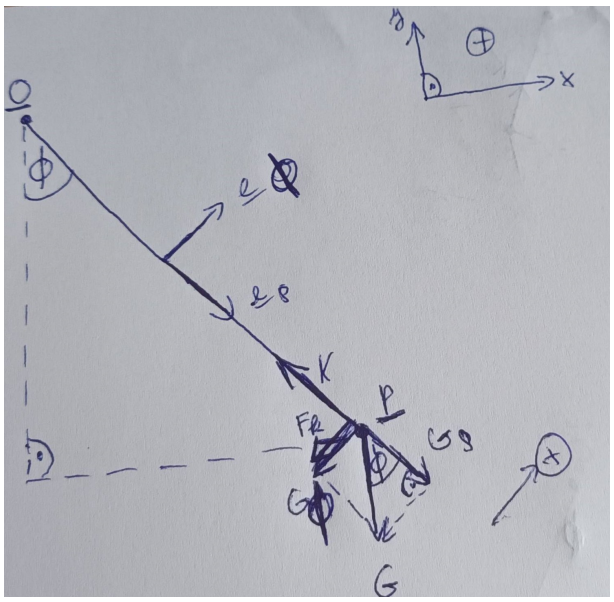
A projekt során az inga szögelfordulásának (ϕ) időszerinti függésének meghatározása volt a cél, ezzel leírva a mozgását. Ebből az egy paraméterből a mozgás többi időtől függő paraméterei is származtatható, erről a fizikai kép leírásánál fog szó esni. A program először egy adatok.txt nevű fájlból bekéri a kezdő paramétereket (fonál hossza, inga tömege, közegellenállási tényező, kezdeti szögelfordulás), majd pedig a mozgását leíró differenciálegyenlet numerikus megoldásából, rekurzióval megkapjuk a szögelfordulás adatokat adott t időpillanatban, amelyeket egy Plot nevű osztály segítségével ábrázoltatjuk egy szabványméretű konzolon. A projekt felépítése során törekedtem arra, hogy átlátható, valamint a későbbiekben módosítható legyen, amennyiben további ötletek támadnának fel a bővítéssel (pl. inga mozgásának ábrázolása egy ablakban SDL külső könyvtár segítségével) kapcsolatban.

4. A felhasznált fizikai modell, számítás menete

4.1. Az alapvető mozgást leíró modell

A fizikai modellt, a mozgást leíró egyenletet alapvetően a Newtoni mechanikából ismert elemek segítségével számoltam ki, azonban amennyiben unatkozna az ember, akkor a módosított Langrange-formalizmus segítségével is kihozható, hasonlóan, a mozgást leíró differenciálegyenlet.

Modellezzük az ingát egy pontszerű testnek, amely egy fonálon függ. Tegyük fel, hogy az inga alapvetően ϕ szöget zár be az y-tengellyel, ekkor az ingára az 1. ábrán látható módon hatnak az erők. Mivel a rendszer henger-szimmetrikus, ezért hengerkoordináta-rendszerbe, jelen esetben (ρ, ϕ) rendszerben vizsgálva írom le a mozgást.



1. ábra. A vizsgált rendszerben ható erők.

Ebből a koordinátarendszerből nézve, a ρ irányba a nehézségi erő ρ komponense (G_ρ) és a kötélerő (K) hat, még ϕ irányba hat a közegellenállási erő (F_k), ami kis sebesség miatt csak a sebességgel arányos (jelen esetben a tangenciális sebességgel [v_T]), és a nehézségi erő ϕ komponense G_ϕ. Használjuk fel, hogy az O pontból vizsgálva a P pontot, az ott ható erőknek forgatónyomatékuk van. Ekkor írjuk fel a forgómozgás alapegyenletét, és az eredő forgatónyomatékot:

$$\theta \cdot \beta = \sum_i M_i = -(M_{G_\phi} + M_{F_k}), \tag{1}$$

ahol β a szöggyorsulás, amely a szögelfordulás második deriváltja, theta a tehetetlenségi nyomaték, M_i pedig az i. forgatónyomaték vektor nagysága, a negatív előjel pedig a felvett pozitív irány miatt adódik. Felhasználva ezeket, és behelyettesítve a forgatónyomatékokat az (1) egyenletbe, az alábbi differenciálegyenletet kapjuk:

$$m \cdot L^2 \cdot \ddot{\phi} = -m \cdot g \cdot sin(\phi) \cdot L - k \cdot v_T \cdot L. \tag{2}$$

Rendezve a (2) egyenletet, és felhasználva, hogy $v_T = \omega \cdot L = \dot{\phi} \cdot L$:

$$\ddot{\phi} + \frac{k}{m}\dot{\phi} + \frac{g}{L} \cdot sin(\phi) = 0, \tag{3}$$

amely egy homogén, nem lineáris, másodrendű differenciálegyenlet, amelynek nincs analitikus, zárt megoldása, ezért a differenciálegyenletet muszáj numerikusan megoldani, amihez az explicit Euler-módszert fogom felhasználni.

Az explicit Euler módszer lényege az, hogy a deriváltat a definíciójával közelítjük, vagyis a deriváltak a következőképpen adódnak:

$$\dot{\phi}(t) \approx \frac{\phi(t + \Delta t) - \phi(t)}{\Delta t} \tag{4}$$

$$\ddot{\phi}(t) \approx \frac{\dot{\phi}(t + \Delta t) - \dot{\phi}(t)}{\Delta t} \tag{5}$$

ami egy megfelelő közelítés, amennyiben Δt megfelelően kicsi, melyre figyeltem a program elkészítése során. Felhasználva az előbb említett lépéseket a (4) és (5) egyenletben, megfelelő átalakításokkal a következő egyenletrendszerre juthatunk:

$$\dot{\phi}(t + \Delta t) \approx \left(1 - \frac{k}{m} \cdot \Delta t\right) \cdot \dot{\phi}(t) - \frac{g}{L} \cdot sin(\phi(t)) \cdot \Delta t, \tag{6}$$

$$\phi(t + \Delta t) \approx \phi(t) + \dot{\phi}(t) \cdot \Delta t. \tag{7}$$

Ez az egyenletrendszer megoldható numerikusan rekurzióval t = 0, Δt, ..., n·Δt ($n \in \mathbb{Z}^+$) időpontokban, amennyiben tudjuk t=0 időpontban $\phi(0)$-t, és $\dot{\phi}(0)$-t.

4.2. Az ütközés modellje

Valós ütközések esetében a relatív sebességeket egyfajta együtthatóval jellemezhetjük, ami a rugalmassági együttható, mely a következőképpen írható fel:

$$e = \frac{v_{rel}^{\ddot{u},u}}{v_{rel}^{\ddot{u},e}}, \tag{8}$$

ahol $v_{rel}^{\ddot{u},e}$ az ütközés előtti, $v_{rel}^{\ddot{u},u}$ az ütközés utáni relatív sebesség a két objektum között. Amennyiben felírjuk a lendületmegmaradást, és felhasználjuk ezt az együttható, valamint, hogy $v = \omega \cdot L = \dot{\phi} \cdot L$, akkor a következő formulákra juthatunk:

$$\dot{\phi}_{1,új} = \frac{m_1 \cdot \dot{\phi}_{1,régi} + m_2 \cdot \dot{\phi}_{2,régi} + m_2 \cdot e \cdot (\dot{\phi}_{2,régi} - \dot{\phi}_{1,régi})}{m_1 + m_2}, \tag{9}$$

$$\dot{\phi}_{2,új} = \frac{m_1 \cdot \dot{\phi}_{1,régi} + m_2 \cdot \dot{\phi}_{2,régi} + m_1 \cdot e \cdot (\dot{\phi}_{1,régi} - \dot{\phi}_{2,régi})}{m_1 + m_2}. \tag{10}$$

5. A projekt felépítése, modulokra bontás

A projekt felépítése során törekedtem az egy osztály - egy modul alapelv betartása mellett, ami szükséges a jobb átláthatóság, valamint a könnyebb szerkeszthetőség érdekében, arra, hogy a C++ alapvető könyvtárain kívűl mást ne használjak. A projekt az alábbi modulokra/csoportokra bonthatóak fel:

- `Coords osztály:` Coords.cpp + Coords.h
- `Dpen osztály` dpen.cpp + dpen.h
- `Inga osztály`: inga.cpp + inga.h
- `Plot osztály`: plot.cpp + plot.h
- `Main:` main.cpp

A modulokra bontás célja az átláthatóság, valamint a könnyebb szerkeszthetőség. A header fájlokba kerültek az osztályok, deklarációk, még a .cpp fájlokba, kivétel a main.cpp, a hosszabb/nagyobb tagfüggvények definiálása, ezekről a továbbiakban olvashatunk. A main.cpp főcélja az osztályok lehetőségeinek bemutatása, valamint a szimuláció elvégzése és a hibakezelés. A továbbiakban az osztályok pontosabb bemutatása következik, logikai sorrendben.

5.1. Coords osztály

A header fájlban található a Coords osztály és a tagfüggvényeinek, tagváltozóinak deklarációja. Az osztály alapvető célja pedig az iterációhoz szükséges paraméterek tárolása. Ez az osztály tulajdonképpen egy speciális vector osztályként funkcionál.

A fájl első sorában látható a `struct xypt(double x, y, phi, phidot, t)` struktúra deklarációja, amely az adatok megfelelő tárolásának megoldása céljából jött létre, a szükséges változókat tárolja el. Ennek a struktúrának az az alapvető célja, hogy ebből egy dinamikus tömböt készíthessünk.

A Coords osztály az alábbi privát tagváltozókkal rendelkezik:

- `xypt *data` - dinamikus tömb, amelyben az iterációs adatokat fogjuk tárolni.
- `int elnum` - a tömb mérete egész szám formájában, ami az adatok biztonságos hozzáadásához, valamint a tömb méretének lekérdezéséhez volt szükséges.

Továbbá, a Coords osztály publikus elemei:

- `Coords()` - a Coords osztály konstruktora, egy üres tömböt hoz létre, elemszám 0 lesz.
- ∼`Coords()` - a Coords osztály destruktora, ami a memóriaszívárgás elkerülése miatt jött létre, kitörli a dinamikus tömböt.
- `Coords(const Coords)` - copy konstruktor, szükséges volt ahhoz, hogy más Coords értékeket fel lehessen használni könnyebben.
- `const Coords& operator=(const Coords)` - = operátor, ami könnyebbé tette az egyik Coordsból a másik Coordsba történő átmásolást, ezzel csökkentve a szükséges kiírt sorszámokat.
- `xypt& operator[](int)` - az osztályban lévő data dinamikus tömb n. elemét adja vissza, ami szükséges volt ahhoz, hogy a későbbiekben könnyebb legyen a másolás.
- `const xypt& operator[](int) const` - ugyanazt csinálja, mint az előbbi, az oka is hasonló, viszont ez azért kellett, ha a [] operátort másképpen is szerettem volna használni.
- `int size() const` - visszaadja a dinamikus tömb méretét, másolásnál előnyös ezt lekérni.
- `void clear()` - lenullázza az elemszámot, a dinamikus tömböt pedig NULL-al teszi egyenlővé. Szükséges volt, hogy véletlen se legyenek hamis adatok benne.
- `void push_back(const xypt&)` - Szokásos push_back függvény. A kapott adatot beszúrja a tömb végére úgy, hogy létrehoz egy elnum+1 méretű tömböt, majd átmásolja a dinamikus tömb elemeit egy új temp tömbbe, újralétrehozza a data dinamikus tömböt, átmásolja annak az elemeit a temp tömbbe, és végül memóriaszívárgás elkerülése végett kitörli a temp tömböt.

5.2. Inga osztály

Az inga osztály főcélja az inga mozgásának szimulálása, a szimulációból származó adatok előállítása. A továbbiakban az inga osztály részletes felépítéséről fogok beszélni.

Az inga osztály az alábbi privát változókkal rendelkezik:

- `double phi0` - a kezdeti szögelfordulás nagysága.
- `double L` - a kötél hossza, amelyre a golyó van felfüggesztve.
- `double m` - az inga tömege kötél és golyó tömegével, viszont ha a kötél tömege sokkal kisebb, mint a golyóé, akkor az elhanyagolható.
- `double k` - a közegellenállás tényező, amibe bele van véve a test geometriai jellemzője, valamint a közeg sűrűsége, amit állandónak tekintettünk a folyamat során.

Az inga osztály publikus elemeinél megjegyezendőek, hogy a getterek és a setterek az iterációhoz szükséges, emiatt keletkeztek. Végül pedig az inga publikus elemei a következőek:

- `inga(double phi0, double L, double m, double k)` - az inga osztály konstruktora, a privát változók értékeit állítja be a beérkező adatok értékeire. Hibát dob, amennyiben valamelyik változó nem megfelelő értéket kap.
- `inga(inga&)` - az inga osztály konstruktora, szükséges volt, hiszen felhasználtam a program során különböző célokra.
- `const inga& operator=(const inga&)` - az inga osztály = operátora, ami szükséges volt, hogy a dpen osztályban könnyebb legyen a konstruktor megírása, valamint, mivel máshol is szükségem volt erre az operátorra.
- `double getPhi0() const` - visszaadja a kezdeti szögelfordulás nagyságát.
- `double getL() const` - visszadja a kötél hosszát.
- `double getM() const` - visszaadja az inga rendszer tömegét.
- `double getK() const` - visszaadja a közegellenállási tényező nagyságát.
- `double setPhi0(double)` - beállíthatjuk vele a kezdeti szögelfordulás értékét.
- `double setL(double)` - beállíthatjuk vele az inga hosszát.
- `double setM(double)` - beállíthatjuk vele a rendszer tömegét.
- `double setK(double)` - beállíthatjuk vele a közegellenállási tényező értékét.
- `friend std::istream& operator»(std::istream&, inga&)` - a könnyebb beolvasást teszi lehetővé, szükséges volt a feladat alapvető kiírása miatt elsősorban, de megkönnyíti az adatok bevitelét szabványos bemenetről. Hibát dob, amennyiben valamelyik változó értéke nem megfelelő.

- `friend std::ostream& operator«(std::ostream&, inga&)` - a könnyebb kiírást teszi lehetővé, oka ugyanaz, mint a beolvasás operátornak.
- `double iteracio(xypt&, double)` - itt történik az erőtörvény iterációja a kapott xypt struktúra adataiból, kiszámolja az új értékeket, és berakja a kapott struktúrába.
- `void refine_dt(const xypt, double&)` - itt történik a megfelelő Δt beállítása, hogy az explicit Euler-módszeres közelítés megfelelő pontosságú legyen.
- `Coords ujcoords(double)` - megadott ideig hívja az iterációs tagfüggvényt az előzőleg kapott adatokkal, és eltárolja ezeket az adatokat, hogy később ki lehessen értékelni.
- `void konvert(std::string, int)` - a fájlból való beolvasást végzi el, és a fontosabb stringeket double típusú számmá konvertálja, hogy a további műveleteket el tudjuk végezni. A második, int típusú paramétere azt adja meg, hogy melyik sort olvassa be a fájlból. Nem volt szükséges az ide való implementációja, a mainbe is megfelelő lett volna, viszont így jobb átláthatóságot nyújt a program tesztelő részében.

5.3. Dpen osztály

Az osztály lényege, hogy két inga mozgását szimulálja úgy, hogy a két inga felfüggesztésének pontja ugyanabban a magasságban vannak, és az ingák ütközését is figyelembe veszi.

A dpen osztály privát változói:

- `inga inga1, inga2` - a két inga adatait tároló, és azokat iterálni képes inga osztályú változók.
- `double e` - a rugalmassági együttható értéke.

A dpen osztály publikus elemei pedig a következőek:

- `dpen(const inga, const inga, double)` - a dpen osztály konstruktora, a privát tagváltozóinak az adatait állítja be a bejövő adatokra. Amennyiben a rugalmassági együttható értéke nem megfelelő, akkor hibát dob.
- `void ujcoords(double, Coords&, Coords&)` - A két inga együttes mozgását iterálja és szimulálja egy megadott ideig, figyelembe veszi az ingák ütközése során bekövetkező energiaváltozást.
- `void ref_dt(const xypt, const xypt, double)` - az előzőekben látott Δt értékét pontosítja úgy, hogy megnézi mindkét inga esetében, hogy mekkora Δt lenne megfelelő, és a kisebbet állítja be, hogy mindkettő esetében pontos szimulálást/iterálást kapjunk.

5.4. Plot osztály

Az osztály a szabványméretű konzolra való kirajzoltatás céljából jött létre, szükséges volt az ábrázoláshoz, valamint az iterált adatok helyes ellenőrzéséhez. Kapott adatokból kirajzolja a szögelfordulást az idő függvényében, valamint képes az adatokból kapott függvényt egy $w \cdot h$ (w = szélesség, h = magasság) nagyságú tömbben tárolni.

A plot osztály az alábbi privát tagváltozókkal rendelkezik:

- `unsigned w` - az ábra szélességét tárolja el, pixel egységben.
- `unsigned h` - az ábra magasságát tárolja, pixel egységben.
- `bool *pmap` - a grafikon pontjainak tárolására szolgáló tömb, itt lehet beállítani, hogy melyik helyen milyen értéket vesz fel a $\phi(t)$ függvény.

Továbbá, a plot osztály publikus elemei az alábbiak:

- `Plot(signed, signed)` - a plot osztály konstruktora, a bejövő paraméterekből konstruálja az ábrázoláshoz szükséges pmap tömböt, illetve kitörli annak az elemeit.
- `Plot (Plot&)` - a plot osztály copy konstruktora, szükséges volt a másoláshoz, valamint ahhoz, hogy az esetleges hibás használatot. Csak ugyanolyan méretű grafikonokat engedi átmásolni az egyikből a másikba.
- `Plot& operator= (const Plot&)` - a plot osztály = operátora, ami szükséges volt a másolásnál, valamint ahhoz, hogy az esetleges hibás használatot levédje. Ugyanúgy csak ugyanolyan méretű grafikonok másolását engedélyezi.
- `~Plot()` - a plot osztály destruktora, ami a memóriaszívárgás akadályozza meg, kitörli a pmap tömböt, ezzel megakadályozva a szívárgás elkerülését.
- `void phimax_keres(const Coords, double&)` - megkeresi a szögelfordulás értékek közül a maximumot a bejövő adatokból, ami szükséges volt, hogy megfelelően legyen méretezve a grafikon a megadott konzolméreten.
- `void clear()` - a grafikon értékeinek tárolására szolgáló tömb elemeit nullázza ki, törli el, ami szükséges volt, hogy esetlegesen többszörös ábrázolásnál ne csússzanak egybe a grafikonok, ezáltal elrontva az egész ábrázolást.
- `void rajzol() const` - kirajzolja a megadott méretű konzolra a pmap tömbben található elemeket, függetlenül attól, hogy az üres vagy sem.
- `void graph(const Coords)` - elkészíti a megfelelő méretezéssel a szögelfordulás-idő grafikont a pmap tömbben tárolva.