МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по практической работе №1

по дисциплине «Вычислительная математика»

ТЕМА: МЕТОДЫ ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ.

Студенты гр. 0382 Чегодаева Е.А,
Сергеев Д.А,

Преподаватель Лавренов А.В.

Санкт-Петербург
2021

Цель работы.

Изучение методов численного интегрирования. Создание трехмерной динамической картинки, изменяющейся согласно физическим законам.

Задание.

Симуляция твёрдого тела легче воды, падающего в воду или всплывающего из неё. Требуется вычислять объём тела, находящийся в воде, и архимедову силу. Создать 3D модель данного явления, а также применить несколько методов численного интегрирования и сравнить их.

Выполнение работы.

Работа выполнена на языке Python 3 с использованием модуля vpython.

Для реализации моделирования осуществляется создание 3D объектов: плоскости, которая представляются толщу воду(depth), поверхность(surface) воды и дно(bottom), шар(s) — представляющий собой предмет, за поведением которого идёт наблюдение. На созданные объекты наложены соответствующие текстуры и/или цвет => В итоге пользователь получает объёмную модель — представляющую собой динамическую картинки для выполнения поставленной задачи: в зависимости от изначальной позиции тела (шара) относительно поверхности воды (поверхности представленного «Аквариума») - тело либо падает вниз, либо всплывает согласно физическим законам, описанным далее.

В работе используются g=9.8 м/с^2, $\rho_{воды}$=997 кг/м^3, $\rho_{тела}$=300 кг/м^3 (либо любое другое, заданное пользователем).

Также реализованы вспомогательные функции:

- F() — данная функция используется для вычисления силы тяжести или силы Архимеда.

$$F = \rho * V * g$$

- ResForce() — данная функция используется для вычисления силы сопротивления.

$$ResForce = k_c * \rho_{воз} * v^2 * \frac{S}{2}$$

- Volume() — данная функция используется для вычисления объёма тела, погруженного в воду.

$$Volume = \pi * h^2 * \frac{(3 * r - h)}{3}$$

- Vy() — данная функция используется для вычисления скорости тела.

$$Vy = a * t$$

- Sy() — данная функция используется для вычисления расстояния пройденного телом.

$$Sy = v * t$$

Динамика 3D модели реализована посредством покадрового изменения картинки с шагом 1/N (N – задаётся пользователем). На каждом шаге вычисляется ускорение - a в зависимости от положения тела.

Первый случай, тело находится выше воды, тогда ускорение тела вычисляется по второму закону Ньютона, так как в этот момент на тело действуют две силы: сила тяжести и сила сопротивления воздуха.

$$a = \frac{k_c * \rho_{воз} * v^2 * \frac{S}{2} - m * g}{m}$$

Второй случай, тело не полностью погружено в воду, тогда по второму закону Ньютона, вычисляется ускорение тела. В данный момент на тело действуют 3 силы: сила тяжести, сила сопротивления воздуха, и сила Архимеда, которая прикладывается к части тела, погруженное в воду.

$$a = \frac{\pi * h^2 * \frac{(3 * r - h)}{3} * V * \rho_{вод} - k_c * \rho_{вод} * v^2 * \frac{S}{2} - m * g}{m}$$

Третий случай, тело полностью находится в воде, в таком случае ускорение так же вычисляется по второму закону Ньютона, ведь на тело действуют 3 силы: сила Архимеда, сила тяжести и сила сопротивления воды. Далее вычисляется скорость тела, а уже используя скорость, определяется координата тела.

$$a = \frac{\rho_{\text{вод}} * V * g - k_c * \rho_{\text{вод}} * v^2 * \frac{S}{2} - m * g}{m}$$

Затем в программе реализованы методы численного интегрирования, которые используются для аппроксимации вычисления координаты тела для его передвижения. Общие обозначения: s.pos.y – координата y тела, sk – скорость тела, первая производная координаты, a – ускорение тела, первая производная скорости, вторая производная координаты, dt – шаг.

- ✓ Метод Эйлера — одношаговый метод первого порядка. Применяется для вычисления скорости (sk=sk+a*dt) и координаты тела (s.pos.y=s.pos.y+sk*dt). Реализован в функции euler()

$$y' = v$$
$$y'' = v' = a$$

Соответственно:

$$v = v + a * \Delta t$$
$$y = y + v * \Delta t$$

- ✓ Метод средней точки — одношаговый метод второго порядка. Применяется для вычисления координаты тела (s.pos.y =s.pos.y+sk*dt+a*dt^2/2), также применяется для скорости, но так как у ускорение отсутствует вторая производная сопоставим с вычислением скорости по методу Эйлера. Реализован в функции midpoint().

$$y' = v$$
$$y'' = v' = a$$

Соответственно:

$$v = v + a * \Delta t$$
$$y = y + v * \Delta t + \frac{a * \Delta t}{2}$$

- ✓ Аналитический метод — скорость и координата вычисляется по известным классическим формулам. Реализован в функции classic().

$$v = v + a * t$$
$$y = y + v * t$$

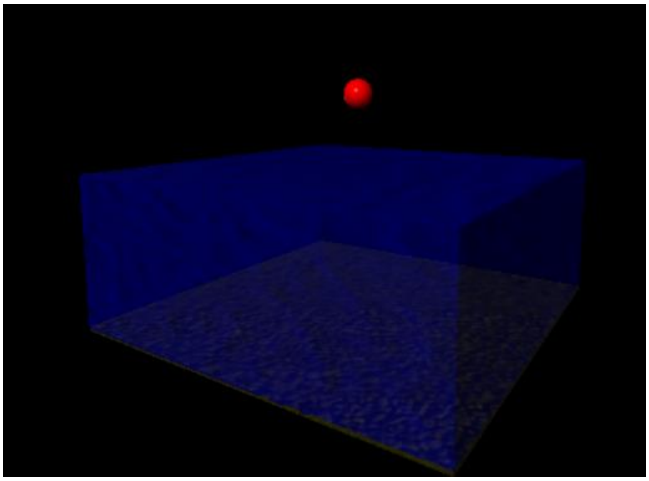
Для проверки корректности интегрирования уравнений реализовано сравнение результатов, полученных аналитическим методом с результатами, полученными посредством методов численного интегрирования.

Исходный программный код смотрите в приложении А.

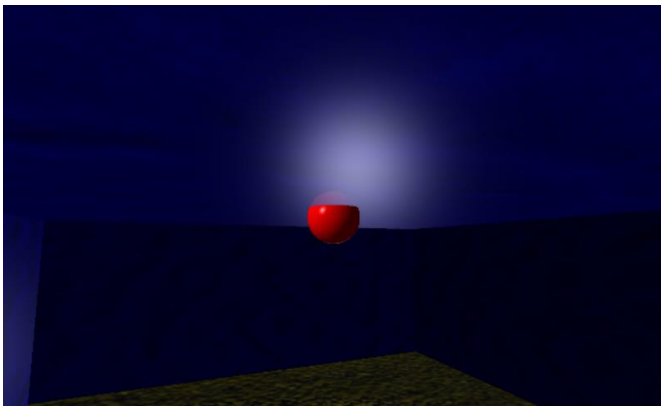
Тестирование.

- Случай, когда тело находится над поверхностью воды:

До:

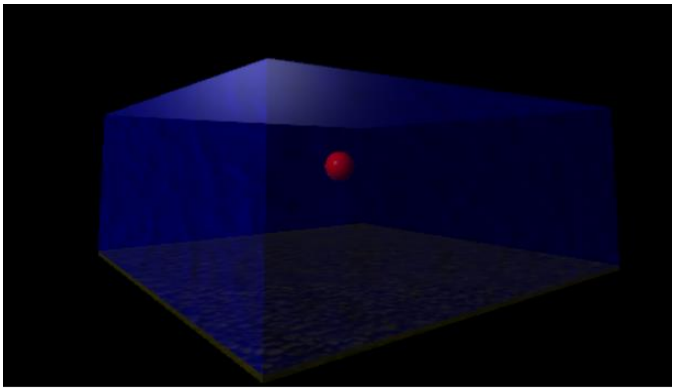


После:

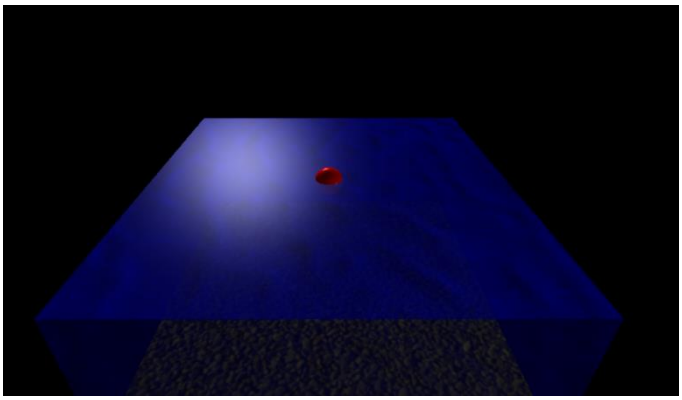


- Случай, когда тело находится под водой:

До:



После:



Выводы.

Были изучены и применены методы численного интегрирования, также была произведена проверка результатов работы методов, которая подтвердила их корректность на данной модели. Создана трёхмерная динамическая картинка тела, падающего или всплывающего из воды, подчиняющегося приложенным к нему физическим силам.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from vpython import *
import math

print("Введите начальную высоту объекта (либо выше уровня воды, либо ниже)")
n=int(input())

#-------------------------------------------------
surface = box(pos=vec(0, 0, 0), size=vec(50, 0.1, 50),
color=color.blue, texture=textures.rough, opacity=0.5)
depth = box(pos=vec(0, -10, 0), size=vec(50, 20, 50),
color=color.blue, texture=textures.rough, opacity=0.4)
bottom = box(pos=vec(0, -20, 0), size=vec(50, 0.5, 50),
color=color.yellow, texture=textures.gravel,opacity=0.4,
shininess=0)
s = sphere(pos=vec(0, 0, 0), radius=2, color=color.red)
s.pos.y=n
ymax = 0
ymin = bottom.pos.y
yT = s.pos.y
g = 9.8
pV = 997
pT=500
v = 4/3*math.pi*s.radius**3
#-------------------------------------------------

def F(p, g, v):
    return p * g * v

def ResForce(v,p,r):
    return 0.47*p*v*v*(math.pi*r**2)/2

def Volume(h,r):
    return math.pi*h**2*(3*r-h)/3

def Vy(a,t):
    return a*t

def Sy(V,t):
    return V*t

def classic():
    N = 50 #кол-во шагов
    dt = 1 / N #шаг
    sk=0.0000000001
    for k in range(1000000*N):
        if (s.pos.y>=ymax+s.radius):
            a=(ResForce(sk,1.2754,s.radius)-F(pT,g,v))/(pT*v)
        else:
            if (s.pos.y>=-s.radius):
                a=(F(pV,g,Volume(s.radius-(s.pos.y),s.radius))-
F(pT,g,v)-sk/math.fabs(sk)*ResForce(sk,997,s.radius))/(pT*v)
```

```
            else:
                a=(F(pV,g,v)-F(pT,g,v)-
sk/math.fabs(sk)*ResForce(sk,997,s.radius))/(pT*v)
        sk+=Vy(a,dt)
        s.pos.y +=Sy(sk,dt)
        rate(100)

def euler():
    sk = 0.0000000000000000001
    N = 50
    dt = 1 / N
    for k in range(100*N):
        if s.pos.y<=ymin+s.radius-bottom.size.y:
            a=0
            sk=0
        else:
            if (s.pos.y>=ymax+s.radius):
                a=(ResForce(sk,1.2754,s.radius)-F(pT,g,v))/(pT*v)
            else:
                if (s.pos.y>=-s.radius):
                    a=(F(pV,g,Volume(s.radius-(s.pos.y),s.radius))-
F(pT,g,v)-sk/math.fabs(sk)*ResForce(sk,997,s.radius))/(pT*v)
                else:
                    a=(F(pV,g,v)-F(pT,g,v)-
sk/math.fabs(sk)*ResForce(sk,997,s.radius))/(pT*v)
        sk+=a*dt
        s.pos.y +=sk*dt
        rate(100)

def midpoint():
    sk = 0.0000000000000000001
    N = 50
    dt = 1 / N
    for k in range(100 * N):
        if s.pos.y <= ymin + s.radius - bottom.size.y:
            a = 0
            sk = 0
        else:
            if (s.pos.y >= ymax + s.radius):
                a = (ResForce(sk, 1.2754,s.radius) - F(pT, g, v)) /
(pT * v)
            else:
                if (s.pos.y >= -s.radius):
                    a = (F(pV, g, Volume(s.radius -
(s.pos.y),s.radius)) - F(pT, g, v) - sk / math.fabs(sk) * ResForce(
                        sk, 997,s.radius)) / (pT * v)
                else:
                    a = (F(pV, g, v) - F(pT, g, v) - sk /
math.fabs(sk) * ResForce(sk, 997,s.radius)) / (pT * v)
        sk+=a*dt
        s.pos.y +=dt*(sk+dt/2*a)
        rate(100)

def sravn():
    s_eu = sphere(pos=vec(-10, 0, 0), radius=2, color=color.green)
    s_mp = sphere(pos=vec(10, 0, 0), radius=2, color=color.yellow)
    s_eu.pos.y = s_mp.pos.y = s.pos.y
    sk_eu=sk_mp=sk = 0.0000000000000000001
```

```
    N = 50
    dt = 1 / N
    for k in range(1000000 * N):
        if s.pos.y <= ymin + s.radius - bottom.size.y:
            a = 0
            sk = 0
        else:
            if (s.pos.y >= ymax + s.radius):
                a = (ResForce(sk, 1.2754, s.radius) - F(pT, g, v))
/ (pT * v)
            else:
                if (s.pos.y >= -s.radius):
                    a = (F(pV, g, Volume(s.radius - (s.pos.y),
s.radius)) - F(pT, g, v) - sk / math.fabs(sk) * ResForce(
                            sk, 997, s.radius)) / (pT * v)
                else:
                     a = (F(pV, g, v) - F(pT, g, v) - sk /
math.fabs(sk) * ResForce(sk, 997, s.radius)) / (pT * v)
            sk += Vy(a, dt)
            s.pos.y += Sy(sk, dt)
            rate(100)

        if s_eu.pos.y <= ymin + s_eu.radius - bottom.size.y:
            a_eu = 0
            sk_eu = 0
        else:
            if (s_eu.pos.y >= ymax + s_eu.radius):
                 a_eu = (ResForce(sk_eu, 1.2754, s_eu.radius) -
F(pT, g, v)) / (pT * v)
            else:
                if (s_eu.pos.y >= -s_eu.radius):
                    a_eu = (F(pV, g, Volume(s_eu.radius -
(s_eu.pos.y), s_eu.radius)) - F(pT, g, v) - sk_eu /
math.fabs(sk_eu) * ResForce(
                            sk_eu, 997, s_eu.radius)) / (pT * v)
                else:
                    a_eu = (F(pV, g, v) - F(pT, g, v) - sk_eu /
math.fabs(sk_eu) * ResForce(sk_eu, 997, s_eu.radius)) / (pT * v)
            sk_eu += a_eu * dt
            s_eu.pos.y += sk_eu * dt
            rate(100)

        if s_mp.pos.y <= ymin + s_mp.radius - bottom.size.y:
            a_mp = 0
            sk_mp = 0
        else:
            if (s_mp.pos.y >= ymax + s_mp.radius):
                a_mp = (ResForce(sk_mp, 1.2754, s_mp.radius) -
F(pT, g, v)) / (pT * v)
            else:
                if (s_mp.pos.y >= -s_mp.radius):
                    a_mp = (F(pV, g, Volume(s_mp.radius -
(s_mp.pos.y), s_mp.radius)) - F(pT, g, v) - sk_mp /
math.fabs(sk_mp) * ResForce(sk_mp, 997, s_mp.radius)) / (pT * v)
                else:
                    a_mp = (F(pV, g, v) - F(pT, g, v) - sk_mp /
math.fabs(sk_mp) * ResForce(sk, 997, s_mp.radius)) / (pT * v)
            sk_mp += a_mp * dt
```

```
            s_mp.pos.y += dt * (sk_mp + dt / 2 * a_mp)
            rate(100)



#euler()
#midpoint()
classic()
#sravn()
```