2年 情報実習⑤,⑥

～剛体の運動と衝突～

**Ｓ２　　　　　　番**

**名前：**

**提出期限**

**年　　　月　　　日**

1. 理論
2. ニュートン力学

ニュートンは，物体の運動に関する性質を法則化し，以下の運動の3法則を提唱した。

**第1法則（ 慣性の法則 ）**

物体は、外部から力が作用しない限り，静止または等速直線運動を続ける。

**第2法則（ ニュートンの運動方程式 ）**

物体の加速度 *a* [m/s2] は，そのとき物体に作用する力 *F* [N] に比例し，物体の質量 *m* [kg] に反比例する。また，その方向は作用する力と同じであり，次のような式で表される。

**第3法則（ 作用・反作用の法則 ）**

２つの物体の間に互いに力が働くとき，それらの力は大きさが等しく，向きは逆である。

1. 万有引力の法則

ニュートンは，この宇宙では全ての物体は互いに重力を及ぼしあっていると提唱した。これを万有引力の法則とよび，物体の質量を*M*および*m* [kg]，互いの距離を*r* [m]，比例定数を*G*としたとき，次の式で表される。

これは，地球上において重力が発生する元となる法則でもある。しかし，地表付近においては複雑な計算となるため，一般に次のような方法で計算される。

1. 自由落下運動

地表付近にある物体は，重力のみが作用する場合ほぼ一定の加速度で落下運動を行う。この加速度を重力加速度*g*とよび， *g* = −9.8 m/s2が使用される場合が一般的である（上向き正とする場合）。なお，月面においては*g* = 1.6 m/s2である。

これに基づいてシミュレーションを行う場合，つぎの２つの方法がある。

* シミュレーション方法①（例1）

自由落下による物体の変位*y* [m]は，物理基礎で学んだ通り，上向きを*y*軸正とすると初速*v*0[m/s]，重力加速度*g* [m/s2]，時刻*t* [s]を使って，

と表される。これはプログラミング言語（Unity C#）で次のように書ける。

|  |
| --- |
| y = v0 \* Time.time + g \* Time.time \* Time.time / 2f; |

y : 変位[m] ，v0 : 初速[m/s] ，g : 重力加速度[m/s2]，Time.time : ゲーム開始からの時間[s]

しかし，この方法には次のような問題がある。

1. ゲーム開始からの経過時間で位置を求めているため，任意の時間から開始できない。
2. シミュレーション中に外力によって速度や位置を変化させることができない。

よって，衝突などさらに複雑なシミュレーションに使用するためには次の方法を用いる必要がある。

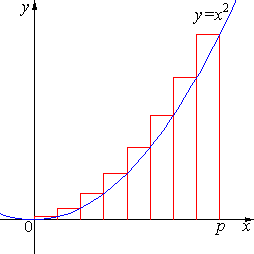
* シミュレーション方法②（例2）

加速度とは「1秒あたりの速度の変化量」であると言える。すなわち，*g* = −9.8とするならば，速さを1秒あたり9.8ずつ減少（負の向きに増加）させれば，自由落下の速さを求めることができるといえる。これはプログラミング言語（Unity C#）で次のように書ける。

|  |
| --- |
| v = v + g \* Time.deltaTime |

v : 速度，g : 重力加速度，Time.deltaTime : 最後のフレームを完了するのに要した時間(秒)

自由落下などの等加速度運動の場合，速度は時間とともに変化する。しかし，シミュレーションを行う場合はフレーム単位でしか演算を行うことができないため，あらゆる時刻における位置や速度を正確に求めることはできない。そこで，フレーム間については右下図のように一定速度として近似する。

次に，速度も同様に「1秒あたりの距離の変化量」であるといえる。これはプログラミング言語（Unity C#）で次のように書ける。

|  |
| --- |
| y = y + v \* Time.deltaTime |

y : 変位，v : 速度， Time.deltaTime : 最後のフレームを完了するのに要した時間(秒)

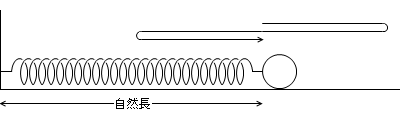
　この方法では，方法①と異なり運動の途中に外力によって速度に変化が生じる場合，vの値を適切に変化させればシミュレーションを継続することができる。

1. ばね振り子

図のように，ばねの先に重りをつけ，振動するようにしたものをばね振り子と呼ぶ。ばねに関する法則として（ フックの法則 ）が一般に知られている。これは，重りにかかる力はばねの伸びた長さに比例するというもので，ばね固有のばね定数を*k*，自然長からの伸び（または縮み）を*x* [m]としたとき，重りにかかる力*F* [N]は次のような式で表せる。

また，より，と式変形すると，加速度a次のように表すことができる。

この式を用いて，重りの動きについてシミュレーションを行う。



* シミュレーション方法

重りは*x*軸上を運動するものとする。自由落下運動のシミュレーションでは加速度は一定の値であったが，ばねのシミュレーションでは加速度は自然長からの伸び*x*（重りの*x*座標とも考えられる）に応じてその都度変化する。そのため，加速度の値を毎フレーム計算し直す必要がある。上で求めた加速度の式は，プログラミング言語（Unity C#）で次のように書ける。

|  |
| --- |
| a = -k \* x / m |

a : 加速度[m/s2]，k : ばね定数，x : 変位[m]，m : 質量[kg]

加速度が決まれば，速度と変位は自由落下シミュレーション②の時と同様の方法で算出することができる。なお，ばね振り子の周期は で表される。

1. 衝突判定（当たり判定）

Unityでは物体同士が衝突や接触をしたかどうかを各種のColliderコンポーネントで検出している。衝突が発生した後の処理は，Unity標準の衝突演算を利用するかどうかで変化する。

* Unity標準の衝突演算を利用する場合

OnCollisionEnter, OnCollisionStay, OnCollisionExitメソッドで衝突を検出できる。この場合，Colliderコンポーネントの「Is Trriger」のチェックが外れている必要がある。

* Unity標準の衝突演算を利用しない場合

OnTriggerEnter, OnTriggerStay, OnTriggerExitメソッドで衝突を検出できる。この場合，Colliderコンポーネントの「Is Trriger」にチェックが入っている必要がある。

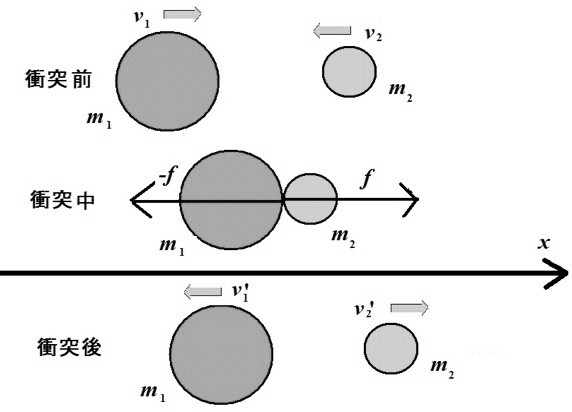
なお，いずれの場合もColliderとともにRigidbodyコンポーネントが必要である。また，「〜Enter」は接触開始時に，「〜Stay」は接触中ずっと，「〜Exit」は離れた時にそれぞれ実行される。

（例3）Rigidbodyコンポーネントで自由落下する物体について，地面（Plane）に衝突した際にConsole画面に「Hit」と表示する

|  |
| --- |
| 1. public class MyScript : MonoBehaviour { 3. void OnCollisionEnter () { 4. print ("Hit"); 5. } 6. } |

（例4）y軸負の向きに1 m/sで進む物体について，地面（Plane）に触れた際にConsole画面に「Hit」と表示する

|  |
| --- |
| 1. public class MyScript : MonoBehaviour { 3. void Update () { 4. this.transform.Translate(0f, -1f \* Time.deltaTime, 0f); 5. } 6. void OnTriggerEnter () { 7. print("Hit"); 8. } 9. } |



1. 衝撃（衝突応答）

衝突が発生した後の物体の運動は（　運動量保存　）の法則に従うとされる。運動量とは質量と速度をかけた値のことで，質量*m*で速度*v*の物体の運動量は*mv*で表される。右図の衝突が起こった際に，衝突の前後で運動量の総和は不変であるとされる。これは次の式で表される。

また，衝突の前後で運動エネルギーが失われなければ，「完全弾性衝突」と呼ばれる。例えば，地面に落としたボールが完全弾性衝突を行えばボールは永久に弾み続ける。逆に，地面に落とした粘土のような，全く弾まない衝突を「完全非弾性衝突」という。現実に起こる衝突は完全弾性と完全非弾性の中間の状態となるため，反発係数*e*を用いて次の式で表される。

この場合，*e* = 1となる衝突を（　完全弾性　）衝突，0 < e < 1 を（　非弾性　）衝突，e = 0を  
（　完全非弾性　）衝突と考えることができる。

反発係数に関する式を変形すると次の式が得られる。

|  |  |
| --- | --- |
|  | …………………① |

また，前述の通り運動量保存の法則は次の式で表される。

|  |  |
| --- | --- |
|  | …………………② |

この2式は衝突後の速度*v*1*'*，*v*2*'*以外は全て数値として与えられるため，二元一次方程式として*v*1*'*，*v*2*'*について解くことができる。

＜解き方＞

式①の両辺に*m*2をかけて式②を足すと，*v*2*'*に関する項が消え次の式が得られる。（①×*m*2＋②）

|  |  |
| --- | --- |
|  | …………………③ |

式①の両辺に*m*1をかけて式②から引くと，*v*1*'*に関する項が消え次の式が得られる。（②－①×*m*1）

|  |  |
| --- | --- |
|  | …………………④ |

* シミュレーション方法：直線上の衝突（例5）

*x*軸上を運動する2つの物体の衝突を考える。衝突のシミュレーションを行うにあたって，まず等速運動を行うためのスクリプトを作成する必要がある。

|  |
| --- |
| 1. public class MyScript : MonoBehaviour { 2. public float v; 3. void FixedUpdate () { 4. this.transform.Translate(v \* Time.deltaTime, 0f, 0f); 5. } 6. } |

また，衝突後の速度に関する式③,④は，プログラミング言語（Unity C#）で次のように書ける。

|  |
| --- |
| v1dash = ((m1-e\*m2)\*v1+(m2+e\*m2)\*v2)/(m1+m2);  v2dash = ((m1+e\*m1)\*v1+(m2-e\*m1)\*v2)/(m1+m2); |

v1 : 自身の衝突前の速度[m/s]，v1dash : 自身の衝突後の速度[m/s]，m1 : 自身の質量[kg]，e : 反発係数  
v2 : 衝突相手の衝突前の速度[m/s]，v2dash : 衝突相手の衝突後の速度[m/s]，m2 : 衝突相手の質量[kg]

ただし，２つの物体のOnTriggerEnterメソッドが衝突時に同時に呼び出され，二重に衝突計算を行ってしまうため，どちらか一方のみが実行されるように（もう一方が実行されないように）プログラム上で仕組みを作る必要がある。

* シミュレーション方法：平面上の衝突

*xy*平面上を運動する2つの物体の衝突を考える。まず等速運動を行うためのスクリプトを*x*, *y*それぞれの向きに対応できるように修正する。

|  |
| --- |
| 1. public class MyScript : MonoBehaviour { 2. public float vx, vy; 3. void FixedUpdate () { 4. this.transform.Translate(vx \* Time.deltaTime, vx \* Time.deltaTime, 0f); 5. } 6. } |

また，衝突に関する演算も同様に修正する。衝突に関する演算は*x*軸方向と*y*軸方向を分けて考え，それぞれ求めればよい。

さらに*z*軸方向を加えると3次元空間での衝突をシミュレートできるが，プログラムが非常に見難くなるため，ベクトルを使って計算するのが一般的である。

1. 実験環境

|  |  |  |  |
| --- | --- | --- | --- |
| 名称/Name | 種別/category | 会社名/Vendor | ﾊﾞｰｼﾞｮﾝ/Ver. |
| macOS | オペレーティングシステム | Apple |  |
| Unity | ゲームエンジン | Unity Technologies |  |

1. 課題　※空気抵抗や摩擦は考えない。また，特に断りがない限り重力加速度は−9.8 m/s2とする。

課題1

Cubeをposition(0, 0, 0)の位置に設置したシーン (Kadai1)について，Cubeを***x*軸**上で振幅  
3 m，1秒間に1往復の頻度で往復運動をさせるスクリプト(MoveBackAndForth)を作成せよ。

なお，以下の機能を実装すること。

①x座標が正のとき物体の色を赤色に，負のとき青色にする。

②開始5秒後に往復運動を停止する。

ヒント：物体の*x*座標はthis.transform.position.x

　　　　開始からの経過時間はTime.time

課題2

Cubeをposition(0, 0, 0)の位置に設置したシーン (Kadai2)について，Cubeをばね振り子の重りと考え，*x*軸上で周期1秒で往復運動するばね振り子の重りの動きをシミュレートせよ。

※ばね振り子を始動させるには，Scene画面上でCubeを*x*軸方向に少し動かせば良い。

ヒント：ばね振り子の周期は計算で求まる。

課題3

Sphereをposition(0, 0, 0)の位置に設置したシーン (Kadai3)について，Sphereに自由落下運動をさせつつ，*x*軸正の向きに1 m/s，*y*軸正の向きに10 m/sの初速を与えて打ち出した物体の運動をシミュレートし，発射地点からの落下地点(発射後y = 0となる地点)までの飛距離をConsole画面に表示せよ。ただし，各パラメータ値の設定はUnityのInspector画面上から行えるようにすること。　ヒント：*x*をで変化させるには x = x + vx \* Time.deltaTimeと書く

課題4

Sphereをposition(0, 0, 0)の位置に設置したシーン (Kadai4)について， Sphereに自由落下運動をさせつつ，発射角（仰角）30°の向きに，初速10 m/sで打ち出した物体の運動をシミュレートし，発射地点からの落下地点までの飛距離（発射後y ≒ 0となる地点でのx座標）を求めよ。ただし，各パラメータ値の設定はUnityのInspector画面上から行えるようにすること。

ヒント：初速を*x*軸成分と*y*軸成分に分けて考えると，  
　　　　*x*軸成分は ，*y*軸成分は となる。

課題5

課題4について発射角を変え，飛距離が最も伸びる発射角を調べ，その時の飛距離を求めよ。

課題6

課題4における飛距離の計測を衝突判定(OnTriggerEnter)を用いた方法で行え。

課題7

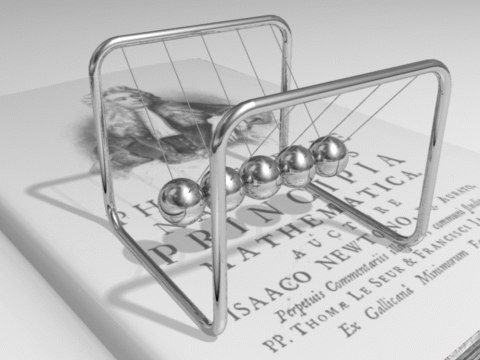
*x*軸上を運動する物体A, Bをそれぞれx = -2, 2の位置に設置し衝突させた。衝突シミュレーションを用いて，次表の空欄に適する値を記入せよ。（e = 1.0）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | 衝突前 | | 衝突後 | |
| Aの質量[kg] | Bの質量[kg] | Aの速度[m/s] | Bの速度[m/s] | Aの速度[m/s] | Bの速度[m/s] |
| 1.0 | 1.0 | 1.0 | -1.0 | -1.0 | 1.0 |
| 1.0 | 2.0 | 1.0 | -1.0 | -1.7 | 0.3 |
| 1.0 | 3.0 | 1.0 | -1.0 | -2 | 0 |
| 1.0 | 4.0 | 1.0 | -1.0 | -2.2 | -0.2 |
| 1.0 | 1.0 | 1.0 | 0.0 | 0.0 | 1.0 |
| 1.0 | 1.0 | 2.0 | 0.0 | 0.0 | 2.0 |
| 1.0 | 1.0 | 2.0 | 1.0 | 1.0 | 2.0 |
| 1.0 | 1.0 | 3.0 | 2.0 | 2.0 | 3.0 |

課題8

*x, y*平面上を運動する物体A, Bをそれぞれ(2, 0)，(-2, 0)の位置に設置し衝突させた。スクリプトを完成させ，衝突シミュレーションを用いて，次表の空欄に適する値を記入せよ。なお，2行目以降は衝突が生じる速度の組み合わせを自ら考え，実験を行うこと。（e = 1.0）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | 衝突前 | | 衝突後 | |
| Aの質量[kg] | Bの質量[kg] | Aの速度[m/s] | Bの速度[m/s] | Aの速度[m/s] | Bの速度[m/s] |
| 1.0 | 1.0 | (1.0, 1.0) | (-1.0, 1.0) | (-1.0, 1.0) | (1.0, 1.0) |
| 1.0 | 1.0 |  |  |  |  |
| 1.0 | 1.0 |  |  |  |  |



課題9

右図のような装置を「ニュートンのゆりかご」という。この装置の動作をシミュレートせよ。

ただし，重力に関するシミュレートは不要で，両端に大質量の物体を配置して反射させるなどの工夫をして再現すること。

考察

Unityは秒間60フレーム程度の頻度で演算を行うが，このような低い頻度の計算で衝突シミュレーションを行う場合，どのような問題が生じるか考えよ。また，その解決策があれば述べよ。