2年 情報実習⑦,⑧

～～

**Ｓ２　　　　　　番**

**名前：**

**提出期限**

**平成　　　年　　　月　　　日**

1. 理論

前回までの実習では，*x*軸上での運動を*xy*平面での運動に拡張するために，*x*軸と*y*軸それぞれ別々に計算を行う方法を使った。これをさらに3次元空間での運動に拡張することを考えたとき，*z*軸を加えた*xyz*空間となるが，*x*座標*y*座標*z*座標を別々に計算するとプログラムの長さが*x*座標だけのものと比べて単純に3倍になってしまい，可読性（プログラムの読みやすさ）が著しく低下する。これを防ぐためにベクトル演算を利用する。

1. ベクトルとは

ベクトルは，高校物理では「向きと大きさをもつ量」として考えるが，計算上は*n*次元ベクトルであれば*n*個の実数の集まりとして扱う。例えば，2次元ベクトルは2つの実数の集まりとして表され，2つの実数を*x*，*y*とすると(*x*, *y*)と表すことができる。これが3次元ベクトルとなれば*xyz*を使って(*x*, *y*, *z*)と表せる。

1. ベクトルの和

2次元ベクトルの場合，ベクトルの和（合成ベクトル）は，幾何学的（図として考える）には平行四辺形の対角線と考えることができる。しかし，計算上はベクトルの各要素の和をそれぞれ求めたに過ぎないため，3次元ベクトルでは次のような式で表せる。

1. ベクトルの差

ベクトルにマイナスが付いた（－1をかけた）場合，大きさは同じで原点対称なベクトルとなる。この性質を使って下図のようにベクトルの差をベクトルの和を使って求めることができる。



図で表すとやや複雑にみえるベクトルの差の求め方であるが，計算上は各要素の差を求めたに過ぎないため，3次元ベクトルでは次のような式で表せる。

（問1）　3次元ベクトル，について，次の各問いの答えをコンピュータを使用して求めよ。なお，Unity C#において3次元ベクトルはVector3型として扱うことができる。

(1) 　　(2) 　　(3)

1. ベクトルの大きさ

ベクトルの大きさは絶対値の記号を使って表し，次の図のように（　三平方　）の定理を使って求めることができる。



また，3次元ベクトルでは次のような式で求めることができる。

大きさ1のベクトルは特別に「（　単位　）ベクトル」と呼び，向きだけを計算に使いたい場合に便利である。

（問2）　3次元ベクトル，について，次の各問いの答えをコンピュータを使用して求めよ。なお，「ベクトルの大きさ」は英語で「magnitude of a vector」となり，Unity C#ではVector3クラスのmagnitudeパラメータでベクトルの大きさを取得できる。

(1) 　　　　　　(2) 　　　　　　(3)

1. ベクトルの積

2つのベクトルの積（掛け算）をベクトルの和と同様に各要素の積として求めた場合，結果は何の意味も持たないベクトルとなってしまう。これは，ベクトルに積の演算が存在しないことを意味する。そこで，ベクトル演算で積の代わりに利用される実数倍，内積，外積の３種類の演算手法を紹介する。

* 1. ベクトルの実数倍

ベクトルに対して，ベクトルではないただの実数（スカラーという）を掛けた場合，掛けた実数の分だけ大きさのみが変化し，向きは変わらない（右図）。

計算上はベクトルの各要素に実数をそれぞれ掛けたに過ぎないため，3次元ベクトルでは次のような式で表せる。

※実数倍は別名「スカラー倍」とも呼ばれる。後述の内積の別名である「スカラー積」と混同しやすいので注意すること。

* 1. 内積

内積はベクトルの各要素の積をすべて合計する計算方法のことである。演算記号として「・」が使用され，計算結果はスカラーとなるため，別名「（　ドット　）積」，「スカラー積」と呼ばれる。3次元ベクトルでは次のような式で表せる。

|  |  |
| --- | --- |
|  | ・・・・・・・・① |

また，内積は2つのベクトルのなす角をθとすると，次のような式でも表すことができる。

|  |  |
| --- | --- |
|  | ・・・・・・・・② |

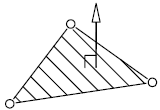
①，②式を使うと，2つのベクトルの相対的な角度を簡単に求めることができるため，あらゆる工業分野において内積の演算は多用される。

より，

※とはの逆関数であるを表している。と書くこともある。

* 1. 外積

外積は内積と同様にベクトルの演算手法のひとつとしてよく使われる。2つのベクトルに対して垂直なベクトルを求めることができるため，面に対する法線ベクトルや，ねじれた位置に生じる力などを求める際によく使われる。なお，演算記号として「×」が使用され，計算結果はベクトルとなるため，別名「（　クロス　）積」，「ベクトル積」と呼ばれる。



法線ベクトル

（問3）　次の各問いの答えをコンピュータを使用して求めよ。なお，Unity C#で内積はVector3クラスのDotメソッド，外積はVector3クラスのCrossメソッドで求めることができる。

(1) ，

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。衝突が発生した場合の処理は，Unity標準の衝突演算を利用するかどうかで変わる。

* Unity標準の衝突演算を利用する場合（Rigidbodyが必要）

OnCollisionEnter, OnCollisionStay, OnCollisionExitが使える。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が使える。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。

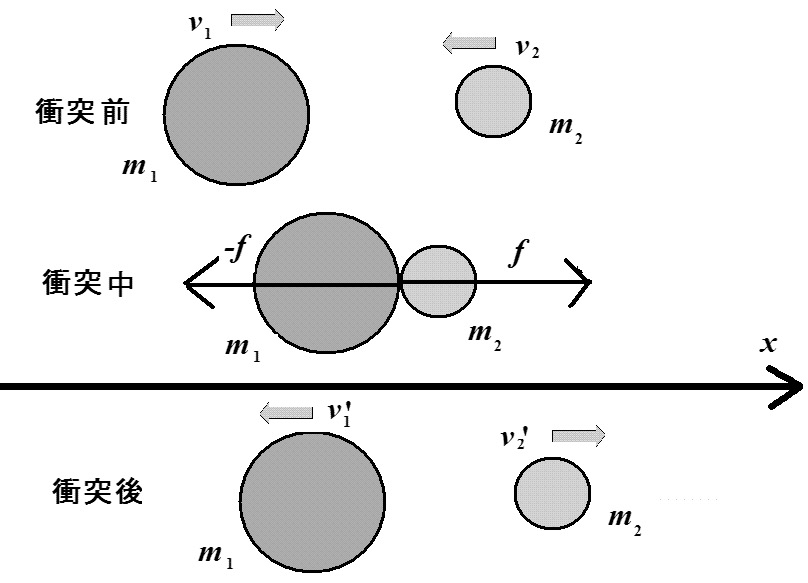
なお，「〜Enter」は接触開始時に，「〜Stay」は接触中ずっと，「〜Exit」は離れた時にそれぞれ実行される。

（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. Debug.Log("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. Debug.Log("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。（①×*m*2＋②）

|  |  |
| --- | --- |
|  | …………………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\*m2)\*v1+(m2+e\*m2)\*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 | 10.12 (Sierra) |
| Unity | ゲームエンジン | Unity Technologies | 2017.1 |

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(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：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

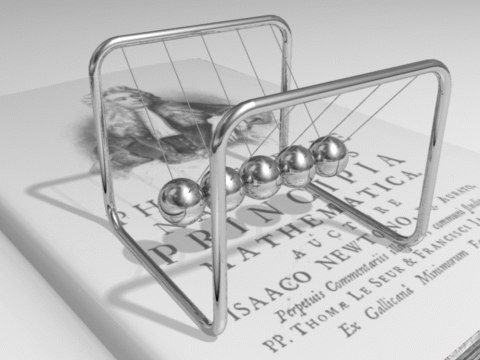
*x*軸上を運動する物体A, Bをそれぞれx = -2, 2の位置に設置し衝突させた。衝突シミュレーションを用いて，次表の空欄に適する値を記入せよ。（e = 1.0）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | 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 | 2.0 | 1.0 | -1.0 |  |  |
| 1.0 | 3.0 | 1.0 | -1.0 |  |  |
| 1.0 | 4.0 | 1.0 | -1.0 |  |  |
| 1.0 | 1.0 | 1.0 | 0.0 |  |  |
| 1.0 | 1.0 | 2.0 | 0.0 |  |  |
| 1.0 | 1.0 | 2.0 | 1.0 |  |  |
| 1.0 | 1.0 | 3.0 | 2.0 |  |  |

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 |  |  |  |  |



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。

考察

①課題7の結果から，衝突の結果一方の物体が静止するための条件を考えよ。

②課題8の結果から，2つの物体が平面上で衝突するための速度に関する条件を考えよ。

※①，②の考察が困難な場合のみ，次の③について考察せよ。

③Unity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。