2年 情報実習⑦,⑧

～物理演算と位置ベクトル～

**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，向きだけを表したい等の場合に使われる。

（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 |

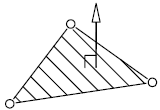
①，②式を使うと，2つのベクトルの相対的な角度を簡単に求めることができるため，あらゆる工業分野において内積の演算は多用される。

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

また，次の式が成り立つ。

内積と合わせることで，180°を超える角度でも正しく求めることができる。

（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.Dotメソッド，外積はVector3.Crossメソッドで求めることができる。

(1) ，のとき，

(2) ，のとき，とのなす角

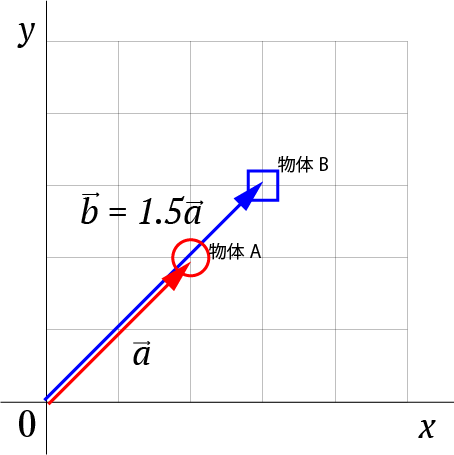
　 ※弧度法→度数法の変換はMathf.Rad2Degを掛ける。（度数法→弧度法はMathf.Deg2Rad）

(3) ，のとき，

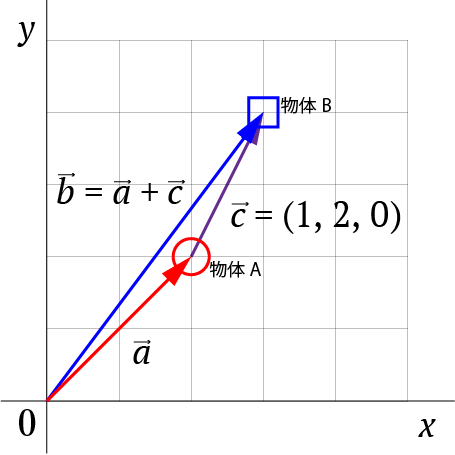
1. Unityにおけるベクトル

Unityでは物体の「位置」は3次元ベクトル型であるVector3型として扱われている。これは，物体の位置を原点からのベクトル（これを「（　位置ベクトル　）」と呼ぶ）であると考えて，直接ベクトル演算で利用できることを意味する。

（例1）原点から物体Aに対しての距離の1.5倍となる位置に物体Bを配置するスクリプト（MoveObject1）を作る。



（例2）物体Aから(*x*, *y*, *z*) = (1, 2, 0)離れた位置に物体Bを配置するスクリプト（MoveObject2）を作る。



1. ベクトルを使った物理シミュレーション

* ベクトルを使った自由落下運動のシミュレーション

自由落下運動のシミュレーションは以前学んだ通り，「加速度は1秒あたりの速度の変化量」，「距離（変位）は1秒あたりの距離の変化量」であると考えプログラムを作成した。このとき，加速度，速度，変位はすべてfloat型（実数型）として扱っていた。

加速度，速度，変位を3次元ベクトルと考え，*xyz*座標をまとめて処理するには次のようにプログラムする必要がある。

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

v : 速度[m/s]（Vector3型），g : 重力加速度[m/s2]（Vector3型），s : 変位[m]（Vector3型），  
Time.deltaTime : 最後のフレームを完了するのに要した時間[秒]

また，速度も同様に「1秒あたりの距離の変化量」であると考え次の計算で距離（変位）を求めた。

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

|  |
| --- |
| 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。

* バネ振り子のシミュレーション

*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。

* 万有引力のシミュレーション

*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

課題1

O(0, 0, 0)，A(1, 0, 0)，B(0, 1, 0)の3点を各頂点とする三角形がある。辺ABの長さをConsole画面に表示するスクリプト（Measure）を作れ。ただし，スクリプトは物体Oのコンポーネントとすること。

Hint : AからBへのベクトル（BからAでも可）の大きさを求めればよい。

課題2

物体Aと原点との中点に物体Bを配置するためのスクリプト（Centering1）を作れ。ただし，スクリプトは物体Bのコンポーネントとすること。

Hint : 物体Aに対しての距離の半分の位置に物体Bを配置すると考える。

課題3

物体Aと物体Bとの中点に物体Cを配置するためのスクリプト（Centering2）を作れ。ただし，スクリプトは物体Cのコンポーネントとすること。

課題4

正四面体の各頂点をA(0, 0, -2)，B(2, 2, -2)，C(0, 2, 0) ，D(2, 0, 0)とする。この正四面体の重心の位置に物体Gを移動するスクリプト(CenterOfGravity)を作れ。ただし，スクリプトは物体Gのコンポーネントとすること。ベクトルを使った重心の求め方はインターネットを利用して調べよ。

課題5

空間上の4点（A, B, C, D）が同一平面上にある場合「OK」，ない場合「NG」とConsole画面に表示するプログラム（Conplanar）を作れ。ただし，スクリプトは物体Aのコンポーネントとすること。

Hint : 4点から任意の3点を選び，その3点の内1点から他の点に向かう2つのベクトルの外積を求める。また，別の選び方で3点を選び同様に外積を求める。外積で求まった2つのベクトルのなす角を調べ，非常に小さい場合（0.0000001radなど適当に決める）同一平面上にあると考えてよい。

課題6

空間に定点A，Bと*z*軸上を自由に移動する動点Pがある。APBの大きさをConsole画面に表示するスクリプト（Shinshu）を作成し，の最大値と，そのときのの値を近似値で求めよ。（信州大 改題）

課題7

４つの物体が同一平面上にあるかを３点の外積の単位ベクトルが等しいかどうかで判定する。

課題8

考察