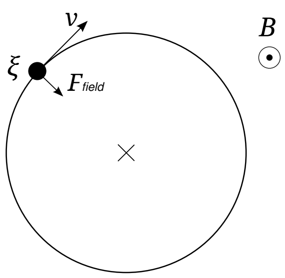
Töltés mozgása a rá merőleges mágneses térben, súrlódással

Vörös Asztrik

Kivonat

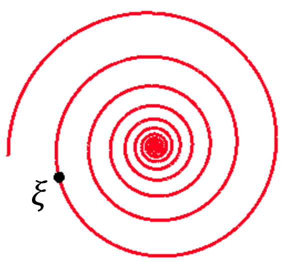
Egy töltött részecske mozgását vizsgáltam a sebességére merőleges mágneses mezőben, ahol sebességtől lineárisan függő súrlódási erő hat rá, de nehézségi erő nem. A pályát egy adott, a mágneses mezőben szereplő kezdő pozíciótól kezdve határoztam meg egészen addig, amíg abból ki nem lép vagy egy szint alá nem lassul a sebessége. A problémát egyenletekkel válaszoltam meg, illetve a kilépést program segítségével állapítottam meg. Végül összevetettem a szimulált eredménnyel a precíz képletet.

1. Súrlódás nélkül



Mivel a mágneses mező vektora ($\vec{B}$) merőleges a töltött részecske ($\xi := (m, q, \vec{v_0})$) sebességre ($\vec{v}$), így a mező ereje ($\vec{F_{\text{field}}}$) síkban nézve merőleges lesz a sebességre. Ez a súrlódástól eltekintve körmozgást fog eredményezni, hiszen infinitezimálisan vizsgálva az erő csak a vektor irányát képes változtatni, a nagyságát nem, valamint a vektor irányának változásával a rá ható erő is vele együtt azonos irányba változik.

2. Súrlódást bevezetve



Ezen problémában egy a sebességtől lineárisan függő súrlódást (F_{fric}) vezetünk be, így megjelenik a körmozgási probláma során ismert tangenciális erő, amely már hatással van a $\vec{v}$ nagyságára. Ebből egyrészt látszódik, hogy $\vec{v}$ függ az időtől (t), illetve mivel F_{fric} függ a sebességtől, így F_{fric} is függeni fog az időtől. Vezessük be a következő egyenletet a súrlódási erőre: $F_{\text{fric}}(t) := -\gamma v(t)$, ahol γ egy pozitív együttható.

Mivel F_{field} a körmozgásért felelős, így

$$\begin{aligned}
F_{\text{field}}(t) &= m\frac{v(t)^2}{r} \\
qv(t)B\sin(90\,\text{deg}) &= m\frac{v(t)^2}{r} \\
r &= \frac{m}{qB}v(t) \\
r(t) &:= \frac{m}{qB}v(t)
\end{aligned}$$

Átrendezéssel látható, hogy a sugár is egy időtől függő tényező. Mivel a körmozgás feltételei a mozgás során teljesülnek, valamint a sugár folyamatosan csökken a sebesség csökkenése miatt, ezért egy spirálszerű mozgást kapunk. Hogy igazoljam feltevésemet, numerikus analízis segítségével közelítőleg ellenőrztem érvelésem 64 bites lebegő pontos számábrázolással, így az fenti képet kaptam, amely megerősítést tesz az eddigiekben.

3. Sebesség meghatározása

A súrlódás egyenletében észrevehetjük az alábbi elsőfokú szeparábilis lineáris differenciálegyenletet.

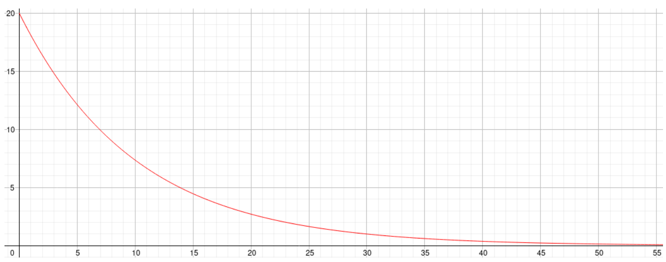
$$\begin{aligned}
F_s &= -\gamma v(t) \\
m\dot{v} &= -\gamma v(t) \\
\dot{v} &= \underbrace{-\frac{\gamma}{m}}_{f(t)}\ \underbrace{v(t)}_{g(v(t))}
\end{aligned}$$

Ennek egyensúlyi helyzetét ($g(v) \equiv 0$) a képletek végén kezeljük. Ezért tegyük fel, hogy $g(v) \neq 0$ és osszunk le vele.

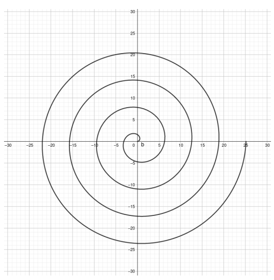
$$\int \frac{v'(t)}{g(v(t))}\mathrm{d}t = \int f(t)\mathrm{d}t$$
$$\int \frac{v'(t)}{v(t)}\mathrm{d}t = \int -\frac{\gamma}{m}\mathrm{d}t$$
$$\ln|v(t)| = -\frac{\gamma}{m}t + c$$
$$|v(t)| = \exp\left(-\frac{\gamma}{m}t + c\right)$$
$$= \exp\left(-\frac{\gamma}{m}t\right) \cdot \exp(c)$$
$$= \exp\left(-\frac{\gamma}{m}t\right) \cdot c$$
$$v(t) = \exp\left(-\frac{\gamma}{m}t\right) \cdot c$$

Láthatjuk, hogy ha $c = 0$, akkor az egyensúlyi helyzetet kaptuk vissza, ezért ez az általános képletünk. Most oldjuk, meg a Cauchy problémát, miszerint $v(0) = v_0$.

$$v(0) = v_0$$
$$\exp(-\frac{\gamma}{m} \cdot 0) \cdot c = v_0$$
$$c = v_0$$



4. Paraméteres görbe



Egy általam ismert paraméteres görbe az $(\alpha\cos(\alpha), \alpha\sin(\alpha))$, ahol megszabhatjuk α-nak az értelmezési tartományát. Láthatjuk, hogy ebben az alakban könnyedén le tudjuk írni a pályát.
A feladatunkhoz általánosítanunk kell a képletet és az idő szerint kifejeznünk a szöget, hiszen eddigi képleteinknek ez a paramétere. Feladatunkhoz a következő általánosítás megfelelő.

$$\begin{pmatrix} O_x + r(t) \cdot cos(\alpha_0 \pm \int_0^t \omega(t)\mathrm{d}t)) \\ O_y + r(t) \cdot sin(\alpha_0 \pm \int_0^t \omega(t)\mathrm{d}t)) \end{pmatrix}$$

4.1. Középpont

Ezzel tudjuk az alakzatot az origóból eltolni. Ehhez vesszük a kezdeti pozíciót ((x_0, y_0)) és az abból kiinduló mező erejét, amit a kezdeti sugár méretére skálázunk át.

$$O \coloneqq (x_0, y_0) + \vec{F_{\text{field}}}(0)\frac{r(0)}{F_{\text{field}}(0)}$$

4.2. α_0

A paraméteres görbék a polárkoordinátás felírás szerint működnek ($(r\cos(\alpha), r\sin(\alpha))$), ezért meg kell határoznunk az alakzat középontjából számított kezdeti szöget (α_0), hogy a paraméteres görbe $t = 0$ esetén a kezdeti Descartes koordinátát adja meg.
Jelölje $(\mathrm{d}x, \mathrm{d}y)$ az alakzat középpontjából vett kezdeti pozíció koordinátáit.

$$\mathrm{d}x \coloneqq x_0 - O_x$$
$$\mathrm{d}y \coloneqq y_0 - O_y$$

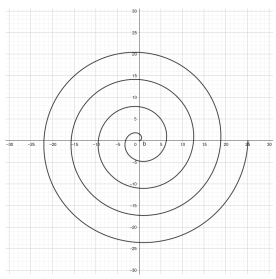
A polár koordinátáról a következő képpen térünk át:

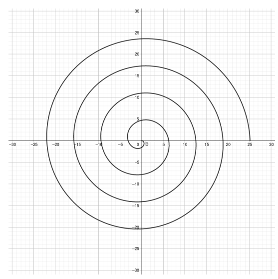
$$r\cos(\alpha_0) = \mathrm{d}x$$
$$r\sin(\alpha_0) = \mathrm{d}y$$

Inverz függvény alkalmazásával nem pontosan kapjuk meg a szöget, hiszen a megfelelő tengelyre tükrözve is ugyanazt az eredményt adják a használt trigonometrikus függvények, azaz azok inverzei nem fedik le a teljes eredeti értelmezési tartományt.
A kettőt együtt felhasználva azonban pontosan megadhatjuk a szöget ($[0, 2\pi)$-n). A *cos* az x tengelyre tükrözött értékeket nem tudja megkülönböztetni, azonban ezt az *arcsin* előjelével meg tudjuk állapítani, hiszen az negatív szöget ad eredményül, ha a paramétere negatív $\Leftrightarrow$ $y < 0 \Leftrightarrow x$ tengely alatt van.

$$\alpha_{0_{\cos}} \coloneqq \arccos\left(\frac{\mathrm{d}x}{r}\right)$$
$$\alpha_{0_{\sin}} \coloneqq \arcsin\left(\frac{\mathrm{d}y}{r}\right)$$
$$\alpha = \begin{cases} \alpha_{0_{\cos}} & \text{ha } \alpha_{0_{\sin}} = 0 \\ \operatorname{sgn}(\alpha_{0_{\sin}}) \cdot \alpha_{0_{\cos}} & \text{egyébként} \end{cases}$$

4.3. Forgási irány ($\pm$)



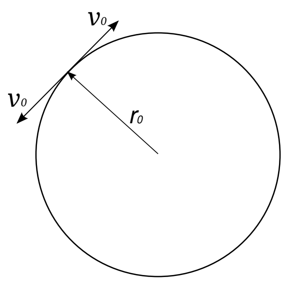


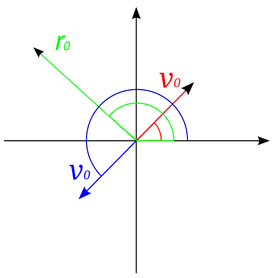
Azt kell megállapítanunk, hogy $v(\vec{0})$ pozitív forgásba indítja el a részecskét vagy negatívba, hiszen ez a folyamat során nem változik. Ezzel ekvivalens probléma ha megállapítjuk, hogy $r(\vec{0})$-hoz képest $v(\vec{0})$ balra vagy jobbra helyezkedik el.

Ehhez egy, a programozási versenyeken használatos geometriai algoritmust használtam.

$$
\begin{aligned}
v_{\vec{\text{from}}} &\coloneqq r(\vec{0}) \\
v_{\vec{\text{to}}} &\coloneqq v(\vec{0}) \\
T &\coloneqq v_{\text{from}y} \cdot v_{\text{to}x} - v_{\text{from}x} \cdot v_{\text{to}y} \\
\text{direction} &= \begin{cases} - & T > 0 \\ + & T < 0 \\ 0 & T = 0 \end{cases}
\end{aligned}
$$

A területből az irányra való következtetés bizonyítása az egyik előjelre (ekvivalens átalakítások):





$$
\begin{aligned}
\alpha_{\text{from}} &> \alpha_{\text{to}} \\
\tan \alpha_{\text{from}} &> \tan \alpha_{\text{to}} \\
\frac{v_{\text{from}y}}{v_{\text{from}x}} &> \frac{v_{\text{to}y}}{v_{\text{to}x}} \\
v_{\text{from}y} v_{\text{to}x} &> v_{\text{to}y} v_{\text{from}x} \\
v_{\text{from}y} v_{\text{to}x} - v_{\text{to}y} v_{\text{from}x} &> 0
\end{aligned}
$$

4.4. $\omega(t)$

Végül az adott pillanatra vonatkozó szögsebességet kell már csak megállapítani a következő összefüggéssel.

$$
\begin{aligned}
\omega(t) &= \frac{v(t)}{r(t)} \\
&= \frac{qB}{m}
\end{aligned}
$$

Láthatjuk, hogy ω nem függ az időtől, így az integrálást könnyen el tudjuk végezni:

$$\int_0^t \omega(t) \mathrm{d}t = \omega t$$

5. Pálya vége

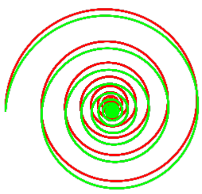
Mivel *exp* sose éri el a 0-át, ezért érdemes egy alsó korlátot bevezeti v-re (v_{bound}).

$$
\begin{aligned}
v(t) &> v_{\text{bound}} \\
\exp(-\frac{\gamma}{m}t) \cdot v_0 &> v_{\text{bound}} \\
\exp(-\frac{\gamma}{m}t) &> \frac{v_{\text{bound}}}{v_0} \\
-\frac{\gamma}{m}t &> \ln\left(\frac{v_{\text{bound}}}{v_0}\right) \\
t &< -\frac{m}{\gamma}\ln\left(\frac{v_{\text{bound}}}{v_0}\right)
\end{aligned}
$$

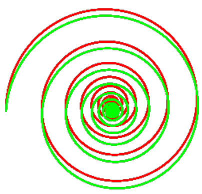
Másik pályát megszakító tényező az az, hogyha a részecske kimegy a mágneses mezőből. Ezt nem precízen, program segítségével állapítottam meg, így ennek pontosság függ az időfelbontásától is.

6. Program

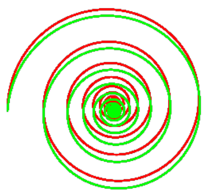
A fentebb említett program *config.json* fájl alapján vezérelhető. Lefutása során képként lementi piros színnel a szimulált pályát és zöld színnel a formulával kapott pályát. Érdemes megfigyelni alacsony időfelbontás mellett is az eltérést.



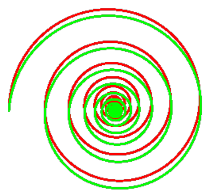
1ms



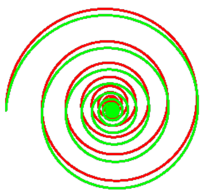
500µs



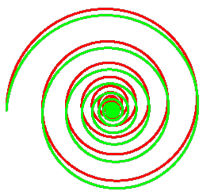
250µs



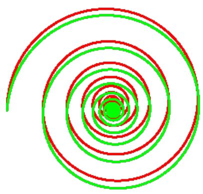
100µs



50µs



25µs



10µs

Szintén kapunk egy formulát, melyet ezen geogebra sémát felhasználva még le is játszhatjuk a mozgást.

Curve((143.333333 + 1.000000/(5.000000*0.300000) * e^(-0.100000/1.000000 * t) * 200.000000 * cos(3.141593 - 1.500000 * t),101.000000 + 1.000000/(5.000000*0.300000) * e^(-0.100000/1.000000 * t) * 200.000000 * sin(3.141593 - 1.500000 * t)),t,0,122.061000)

Az elkövetkezőekben a forráskódot láthatjuk, ahol az elvégzett számítások már le vannak dokumentálva az előbbiekben, így kevésbé szerepelnek kommentek ott, ahol egyértelmű az elvégzett számítás oka.

config.json

```json
{
  "MagneticField": {
    "Value": 0.3,
    "Direction": "Up",
    "Width": 255,
    "Height": 235
  },
  "Particle": {
    "Position": "10 101",
    "Velocity": "0 200",
    "Mass": 1,
    "Charge": 5
  },
  "Friction": {
    "Coefficient": 0.1
  },
  "VelocityBound": 1e-3,
  "TimeDelta": "500us",
  "RAM": 1000
}
```

main.go

```go
package main

import (
  "charge-in-magnetic-field/draw"
  "charge-in-magnetic-field/physics"
  "charge-in-magnetic-field/vector"
  "encoding/json"
  "fmt"
  "log"
  "math"
  "os"
  "time"
)

var config physics.Config
var bottomleft = vector.Create(0, 0)
var topright vector.Vector

//topright exclusive
func inside(pos vector.Vector) bool {
  return pos.X >= bottomleft.X &&
    pos.Y >= bottomleft.Y &&
    pos.X < topright.X &&
    pos.Y < topright.Y
}

func numerical() {
  //step particle with the current values for timeDelta time
  //then draw it
  part := config.Part
  for part.Velocity.Length() > config.VelocityBound && inside(part.Position) {
    draw.Draw(part.Position)

    part.Step(config)
  }
}

func velocityLength(t time.Duration) float64 {
  return math.Exp(-config.Fric.Coefficient/config.Part.Mass*t.Seconds()) * config.Part.Velocity.Length()
}

```

```go
func radiusLength(t time.Duration) float64 {
  return config.Part.Mass / (config.Part.Charge * config.MF.Value) * velocityLength(t)
}

func radius(t time.Duration) vector.Vector {
  F_field := config.MF.Force(config.Part)
  rLength := config.Part.Mass * velocityLength(t) / (config.Part.Charge * config.MF.Value)
  r := vector.Scalar(F_field, rLength/F_field.Length())
  r.Negate()
  return r
}

func center() vector.Vector {
  r0 := radius(0)
  //radius vector goes from its origo
  r0.Negate()
  O := vector.Add(config.Part.Position, r0)
  return O
}

func alpha0() float64 {
  O := center()
  r0 := radiusLength(0)
  dx := config.Part.Position.X - O.X
  dy := config.Part.Position.Y - O.Y
  dxNorm := dx / r0
  dyNorm := dy / r0

  //float error correction
  if dxNorm < -1 {
    dxNorm = -1
  } else if dxNorm > 1 {
    dxNorm = 1
  }
  if dyNorm < -1 {
    dyNorm = -1
  } else if dyNorm > 1 {
    dyNorm = 1
  }

  alpha0cos := math.Acos(dxNorm)
  alpha0sin := math.Asin(dyNorm)
  if alpha0sin == 0 {
    return alpha0cos
  }
  if alpha0sin < 0 {
    return -alpha0cos
  }
  return alpha0cos
}

type Direction int

const (
  Left Direction = iota
  Right
  Straight
)

func direction(base, from, to vector.Vector) Direction {
  from.Sub(base)
  to.Sub(base)
  area := from.Y*to.X - from.X*to.Y
  if area < 0 {
    return Left
  }
  if area > 0 {
    return Right
  }
  return Straight
}

func omega() float64 {
  return config.Part.Charge * config.MF.Value / config.Part.Mass
}

```

```go
func position(t time.Duration, O vector.Vector, w, a0 float64, pm Direction) vector.Vector {
  //parametric equatation
  da := w * t.Seconds()
  if pm == Right {
    da = -da
  }
  posX := O.X + radiusLength(t)*math.Cos(a0+da)
  posY := O.Y + radiusLength(t)*math.Sin(a0+da)
  return vector.Create(posX, posY)
}

func mathematical() {
  //constant values
  O := center()
  a0 := alpha0()
  pm := direction(vector.Create(0, 0), radius(0), config.Part.Velocity)
  if pm == Straight {
    panic("vec is not perpendicular to mf force")
  }
  w := omega()
  tEnd := -config.Part.Mass / config.Fric.Coefficient *
      math.Log(config.VelocityBound/config.Part.Velocity.Length())

  //calculate position for each time to draw it for comparision
  t := time.Duration(0)
  pos := position(t, O, w, a0, pm)
  //always recalculate t as timeDelta is small
  for i := 0; t.Seconds() < tEnd && inside(pos); i++ {
    draw.Draw(pos)

    t = time.Duration(i+1) * config.TimeDelta
    pos = position(t, O, w, a0, pm)
  }

  //Geogebra format for equatation
  dir := "+"
  if pm == Right {
    dir = "-"
  }
  fmt.Print("Curve((")
  fmt.Printf(
    "%f + %f/(%f*%f) * e^(-%f/%f * t) * %f * cos(%f %s %f * t)",
    O.X,
    config.Part.Mass, config.Part.Charge, config.MF.Value,
    config.Fric.Coefficient, config.Part.Mass, config.Part.Velocity.Length(),
    a0, dir, w,
  )
  fmt.Print(",")
  fmt.Printf(
    "%f + %f/(%f*%f) * e^(-%f/%f * t) * %f * sin(%f %s %f * t)",
    O.Y,
    config.Part.Mass, config.Part.Charge, config.MF.Value,
    config.Fric.Coefficient, config.Part.Mass, config.Part.Velocity.Length(),
    a0, dir, w,
  )
  fmt.Print("),")
  fmt.Printf("t,0,%f", t.Seconds())
  fmt.Println(")")
}

func main() {
  //read json
  file, err := os.Open("config.json")
  if err != nil {
    log.Fatalln("can not open")
  }
  if json.NewDecoder(file).Decode(&config) != nil {
    log.Fatalln("can not parse")
  }
  file.Close()
  config.TimeDelta = time.Duration(config.TimeDeltaJSON)

  //check if vec pos on mf
  topright = vector.Create(float64(config.MF.Width), float64(config.MF.Height))
  if !inside(config.Part.Position) {
    panic("starting position is not inside magnetic field")
```

```
	}

	//draw
	draw.Start(config)

	draw.SetRGB(1, 0, 0)
	numerical()

	draw.SetRGB(0, 1, 0)
	mathematical()

	draw.Save()
}
```

vector/vector.go

```
package vector

import (
	"encoding/json"
	"errors"
	"math"
	"strconv"
	"strings"
)

type Vector struct {
	X, Y float64
}

func Create(x, y float64) Vector {
	return Vector{
		X: x,
		Y: y,
	}
}
func (v *Vector) Negate() *Vector {
	v.X = -v.X
	v.Y = -v.Y
	return v
}
func (v *Vector) Scalar(value float64) *Vector {
	v.X *= value
	v.Y *= value
	return v
}
func Scalar(v Vector, value float64) Vector {
	return *v.Scalar(value)
}
func (v *Vector) RotateP90() {
	v.X, v.Y = -v.Y, v.X
}
func (v *Vector) RotateN90() {
	v.X, v.Y = v.Y, -v.X
}
func (v *Vector) Length() float64 {
	return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
func (v *Vector) SetLength(length float64) *Vector {
	return v.Scalar(length / v.Length())
}
func (v *Vector) Add(v2 Vector) *Vector {
	v.X += v2.X
	v.Y += v2.Y
	return v
}
func Add(v1 Vector, v2 Vector) Vector {
	return *v1.Add(v2)
}
func (v *Vector) Sub(v2 Vector) *Vector {
	v.X -= v2.X
	v.Y -= v2.Y
	return v
}
func Sub(v1 Vector, v2 Vector) Vector {
```

```
  return *v1.Sub(v2)
}

//UnmarshalJSON parses string from json to vector
func (v *Vector) UnmarshalJSON(b []byte) error {
  var text string
  if err := json.Unmarshal(b, &text); err != nil {
    return err
  }
  cooS := strings.Split(text, " ")
  if len(cooS) != 2 {
    return errors.New("coordinate should have two values")
  }

  var err error
  if v.X, err = strconv.ParseFloat(cooS[0], 64); err != nil {
    return errors.New("could not parse X")
  }
  if v.Y, err = strconv.ParseFloat(cooS[1], 64); err != nil {
    return errors.New("could not parse Y")
  }
  return nil
}
```

physics/config.go

```
package physics

import (
  "encoding/json"
  "errors"
  "time"
)

type Config struct {
  MF   MagneticField `json:"MagneticField"`
  Part Particle      `json:"Particle"`
  Fric Friction      `json:"Friction"`
  //https://github.com/golang/go/issues/10275
  TimeDeltaJSON Duration `json:"TimeDelta"`
  TimeDelta     time.Duration
  VelocityBound float64 `json:"VelocityBound"`
  //specifies draw.buffer size
  RAM int `json:"RAM"`
}

type Duration time.Duration

//UnmarshalJSON parses string from json to time.Duration
func (d *Duration) UnmarshalJSON(b []byte) error {
  var text string
  if err := json.Unmarshal(b, &text); err != nil {
    return err
  }

  t, err := time.ParseDuration(text)
  if err != nil {
    return errors.New(`a duration string is a possibly signed sequence of decimal numbers, each
      with optional fraction and a unit suffix, such as "300ms", "-1.5h" or "2h45m". Valid time
      units are "ns", "us" (or "µs"), "ms", "s", "m", "h"`)
  }
  *d = Duration(t)

  return nil
}
```

physics/friction.go

```
package physics

import "charge-in-magnetic-field/vector"

type Friction struct {
  //Coefficient should be positive here
```

```
  Coefficient float64 `json:"Coefficient"`
}

func (f *Friction) Force(velocity vector.Vector) vector.Vector {
  return *velocity.Scalar(f.Coefficient).Negate()
}
```

physics/magnetic-field.go

```
package physics

import (
  "charge-in-magnetic-field/vector"
  "encoding/json"
  "errors"
  "strings"
)

type Direction int

const (
  Up Direction = iota
  Down
)

type MagneticField struct {
  Width  uint      `json:"Width"`
  Height uint      `json:"Height"`
  Dir    Direction `json:"Direction"`
  Value  float64   `json:"Value"`
}

func (mf *MagneticField) Force(p Particle) vector.Vector {
  force := p.Velocity
  force.SetLength(p.Charge * p.Velocity.Length() * mf.Value)

  //determine rotation direction
  var positive bool
  if mf.Dir == Up {
    positive = false
  } else {
    positive = true
  }
  if p.Charge < 0 {
    positive = !positive
  }

  if positive {
    force.RotateP90()
  } else {
    force.RotateN90()
  }

  return force
}

//UnmarshalJSON parses string from json to Direction
func (d *Direction) UnmarshalJSON(b []byte) error {
  var text string
  if err := json.Unmarshal(b, &text); err != nil {
    return err
  }
  text = strings.ToLower(text)
  if text == "up" {
    *d = Up
  } else if text == "down" {
    *d = Down
  } else {
    return errors.New("Direction isn't up nor down")
  }
  return nil
}
```

physics/particle.go

```go
package physics

import "charge-in-magnetic-field/vector"

type Particle struct {
  Position vector.Vector `json:"Position"`
  Velocity vector.Vector `json:"Velocity"`
  Mass     float64       `json:"Mass"`
  Charge   float64       `json:"Charge"`
}

//Step uses numerical analysis to calculate particle's next position and velocity
func (p *Particle) Step(config Config) {
  force := vector.Add(config.MF.Force(*p), config.Fric.Force(p.Velocity))
  acceleration := vector.Scalar(force, 1/p.Mass)

  p.Position.Add(vector.Scalar(p.Velocity, config.TimeDelta.Seconds()))
  p.Velocity.Add(vector.Scalar(acceleration, config.TimeDelta.Seconds()))
}
```

draw/draw.go

```go
package draw

import (
  "charge-in-magnetic-field/physics"
  "charge-in-magnetic-field/vector"
  "fmt"
  "log"

  "github.com/fogleman/gg"
)

var config physics.Config
var ctx *gg.Context
var buffer []vector.Vector
var bufferIndex int

func SetRGB(r, g, b float64) {
  //unflushed items would also get this colour so flush
  if bufferIndex != 0 {
    flush()
  }

  ctx.SetRGB(r, g, b)
}
func Draw(v vector.Vector) {
  buffer[bufferIndex] = v
  bufferIndex++

  //buffer may get full
  if bufferIndex == len(buffer) {
    flush()
  }
}
func flush() {
  for i := 0; i < bufferIndex; i++ {
    ctx.DrawPoint(buffer[i].X, float64(config.MF.Height)-buffer[i].Y, 1)
  }
  ctx.Fill()
  bufferIndex = 0
}
func Start(c physics.Config) {
  config = c

  buffer = make([]vector.Vector, config.RAM)
  ctx = gg.NewContext(int(config.MF.Width), int(config.MF.Height))
}
func Save() {
  //buffer may have unflushed points
  if bufferIndex != 0 {
    flush()
  }

```

```go
  if err := ctx.SavePNG(fmt.Sprintf("result-%s.png", config.TimeDelta)); err != nil {
    log.Fatalln(err)
  }
}
```

7. Források

- Koordinátarendszeres képek: Geogebra