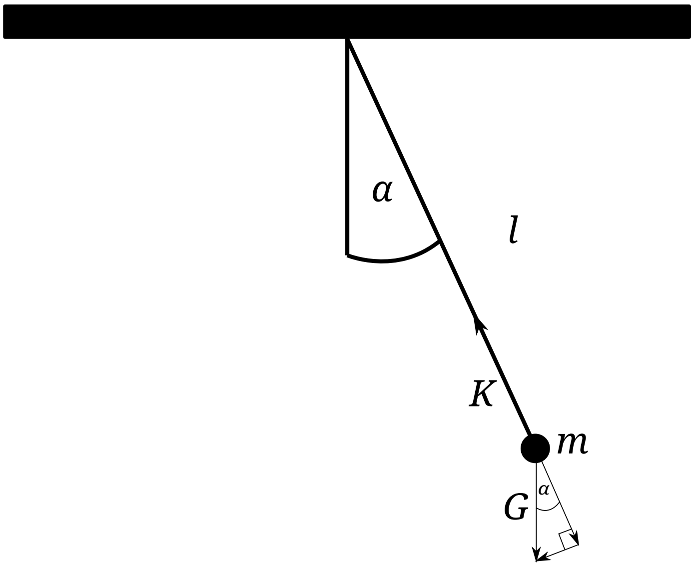
Fonálinga periódusideje nagy szögekre

Vörös Asztrik

Abstract

A modellezés célja, hogy megállapítsuk egy adott hosszúságú inga periódusidejét nagy szögekre is, majd a százalékos eltérést kimutassuk a kis szögekre adott képlethez képest. A metódust felhasználva szintén meghatározhatjuk, hogy adott hibahatáron belül maradáshoz maximum mekkora szöggel téríthetjük ki az ingát.

1 Dinamikai jellemzés



Az inga végén található súlyra felírva az érintő és sugár irányú erőket a következőt kapjuk:
(1) $K - mg\cos\alpha = F_{cp} = ma_{cp} = m\frac{v^2}{l}$
(2) $-mg\sin\alpha = ma_t$
Ahol K a kötélerő, m a felfüggesztett test tömege, g a nehézségi gyorsulás, a_{cp} a centripetális gyorsulás, a_t a tangenciális gyorsulás és végül v a test sebessége. A (2)-es egyenlet negatív előjelét az indokolja, hogy az erő mindig a kitérés csökkenéséhez járul hozzá.
A periódusidő meghatározásához nekünk a (2) egyenletre van szükségünk, hiszen az határozza meg a test pozícióját, míg az (1)-es egyenlet a geometriai viselkedését biztosítja. Átrendezve a következőt kapjuk:
(3) $a_t = \beta l$
(2+3) $\beta = -\frac{g}{l}\sin\alpha$
Ahol β a szöggyorsulás. Ez alapján nem tudjuk alkalmazni a harmonikus rezgőmozgáshoz ismert képletet, mivel a szög második deriváltja a szög szinuszától függ. Mivel tanulmányaim nem terjednek ki addig, hogy ezt a differenciál egyenletet megoldjam, ezért más módszert alkalmaztam.

2 Numerikus analízis - Euler algoritmus

Az Euler módszer kis szélességű téglányi területekkel közelíti meg a függvények alatti területet, amivel a számunkra szükséges szögpozíciót is megkaphatjuk. Az algoritmus a következő:
f_i'' =[állapot alapján kiértékelhető]
$f_i' = f_{i-1}' + f_i''\Delta x$
$f_i = f_{i-1} + f_i'\Delta x$
Ahol $\Delta x \ll 1$.

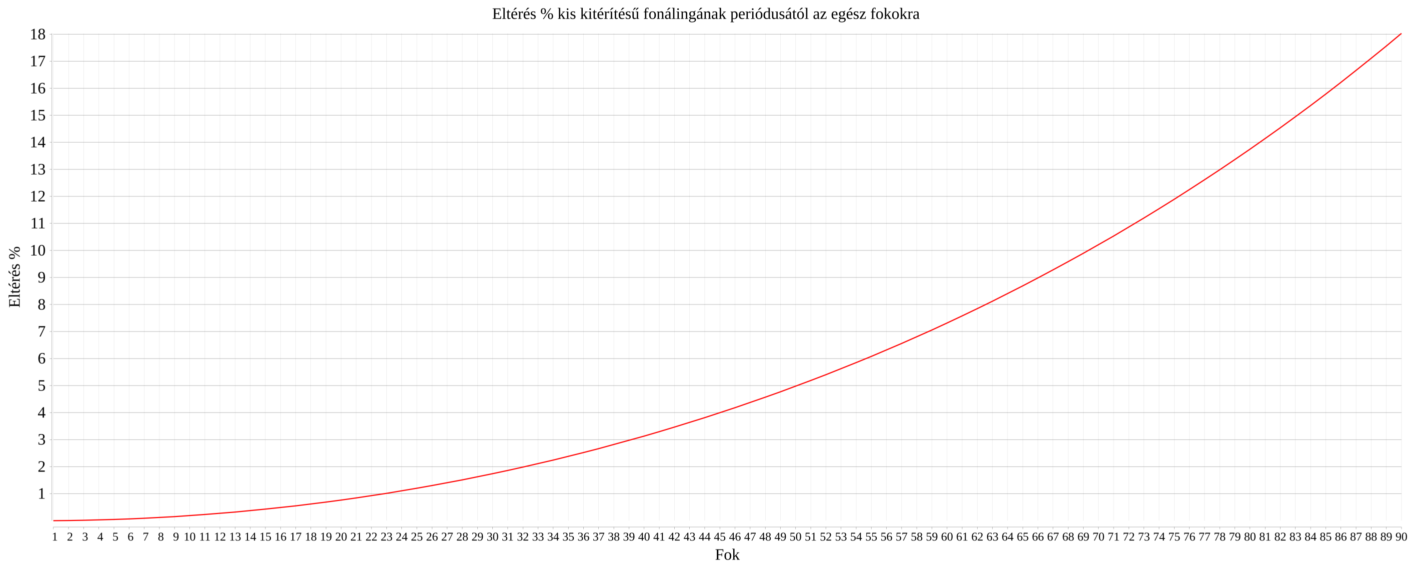
Ezt alkalmazva a problémára:
$\alpha_i'' = \beta_i = -\frac{g}{l}\sin\alpha_{i-1}$
$\alpha_i' = \omega_i = \omega_{i-1} + \beta_i\Delta t$
$\alpha_i = \alpha_{i-1} + \omega_i\Delta t$
A periódusidőt megkaphatjuk, ha megkeressük azt a $t \neq 0$ időpillanatot, ahol a kitérés megegyezik a $t = 0$ időpontban lévővel.

3 Százalékos eltérés

Az alább megadott paraméterek szerint készült a lentebb található grafikon, amely megmutatja az egyszerű fonálinga periódusidejéhez képesti százalékos eltérést minden fokra $\in (0, 90] \cap \mathbb{Z}$:
$l = 1m$
$g = 9.8\frac{m}{s^2}$
$\Delta t = 0.0001s = 100\mu s$
$T_{\alpha\ll 1} = 2\pi\sqrt{\frac{l}{g}}$
Az eredményekből láthatjuk, hogy kis szögekre $\alpha \ll 1$ vonatkozó képletet közel pontos eredményt ad. A grafikon más paraméterekre is ugyanazt az eredményt produkálja, így arra következtethetünk, hogy független azoktól.

4 Hibahatár

Mivel a százalékos eltérés monoton, ezért egy adott százalékhoz tartozó fok megkereséséhez használhatunk bináris keresést.
Az algoritmus lényege a következő: kijelölünk 2 pontot, az egyik határon belül lévő, a másik határon kívül lévő.
Egy lépésben kijelölünk egy harmadik pontot, ami pontosan a kettő átlagán helyezkedik el. Ezután megvizsgáljuk, hogy ez a határon belül vagy kívül van, majd eszerint az elsőnek felvett 2 pont közül a megfelelő értékét átírjuk a 3. pontnak az értékére. Ezt egészen addig csináljuk amíg a 2 pont közötti távolság nem lesz $\leq \Delta\,\text{deg}$.
Mivel a futásidő $\mathcal{O}(\log_2 n)$, így rövid időn belül, nagy pontossággal megkaphatjuk az értéket.
Példának okáért 2%-os hibahatáron belül maradáshoz a kapott érték:
$\Delta\,\text{deg} = 0.000001\,\text{deg}$
$\Delta t = 0.00001s = 10\mu s$
$\alpha = 32.1207920678108669817447500850574006528569...\,\text{deg}$
Azonban figyelembe kell venni a periódus-számító algoritmus pontosságát a megadott Δt-vel, illetve a float szabvány limitációit, így az eredményt érdemesebb az alábbi közelítéssel megadni: $\alpha \approx 32.1\,\text{deg}$



5 Kód - Lényegi rész

```go
package main

import (
  "fmt"
  "math"
  "math/big"

  "gonum.org/v1/plot/plotter"
)

func main() {
  gravity := bigfloatCreate(9.8)
  length := bigfloatCreate(1)

  percentage := bigfloatCreate(2)
  degree := getDegreeLimitForDifferencePercentage(
    gravity,
    length,
    bigfloatCreate(0.000001),
    bigfloatCreate(0.00001),
    percentage,
  )
  fmt.Printf(
    "Maximal degree to be under or equal to %2.5f%% is about %v deg\n",
    percentage,
    degree,
  )
  differencePercentageS := getDifferencePercentageS(
    gravity,
    length,
    bigfloatCreate(1),
    bigfloatCreate(0.0001),
  )
  plotWrite(differencePercentageS, "difference.png")
}

//getDegreeLimitForDifferencePercentage return the maximal degree to not be
//over the given difference percentage between actual and simple pendulum's period
//percentage ∈ [0, inf)
//delta should be small enough that delta's difference percentage is under or equal to percentage
func getDegreeLimitForDifferencePercentage(
  gravity,
  length,
  degreeDelta,
  timeDelta,
  percentage *big.Float,
) *big.Float {
  //at 0 deg both have Inf as period so they have 0% difference which is also the minimum value
```

```
  degreeUnderOrEqual := bigfloatCreate(0)
  //the differnce percentage at 90 deg might be the percentage we want
  degreeOver := bigfloatCreate(0).Add(bigfloatCreate(90), degreeDelta)
  degreeCurrent := bigfloatCreate(0).Add(degreeUnderOrEqual, degreeOver)
  degreeCurrent.Quo(degreeCurrent, bigfloatCreate(2))

  //binary search exact value until in error range
  for bigfloatCreate(0).Add(degreeUnderOrEqual, degreeDelta).Cmp(degreeOver) == -1 {
    period := periodGet(gravity, length, degreeCurrent, timeDelta, bigfloatCreate(0))
    differencePercentage := getDifferencePercentage(gravity, length, period)

    //half the search filed
    if differencePercentage.Cmp(percentage) != 1 {
      degreeUnderOrEqual.Copy(degreeCurrent)
    } else {
      degreeOver.Copy(degreeCurrent)
    }

    //new halfing point
    degreeCurrent = degreeCurrent.Add(degreeUnderOrEqual, degreeOver)
    degreeCurrent.Quo(degreeCurrent, bigfloatCreate(2))
  }

  return degreeCurrent
}

//getDifferencePercentageS returns difference percentage between simple and actual pendulum's period for specified degrees:
//X ∈ [degreeDelta, 90]
//Y ∈ [0, Inf)
func getDifferencePercentageS(gravity, length, degreeDelta, timeDelta *big.Float) plotter.XYs {
  //do not set degreeStart to 0 as plot can not draw infinite value
  degreeStart := degreeDelta
  degreeEnd := bigfloatCreate(90)

  var differencePercentageS plotter.XYs

  i := 0
  for {
    //current degree
    //calculate each time to get exact value
    degree := bigfloatCreate(0).Mul(bigfloatCreate(float64(i)), degreeDelta)
    degree.Add(degree, degreeStart)

    //degree may not be equal to degreeEnd either because
    //degreeDelta's value or
    //float precision
    if degree.Cmp(degreeEnd) == 1 {
      break
    }

    period := periodGet(gravity, length, degree, timeDelta, bigfloatCreate(0))
    percentageDifference := getDifferencePercentage(gravity, length, period)

    //save data for plot
    degree64, _ := degree.Float64()
    percentageDifference64, _ := percentageDifference.Float64()
    differencePercentageS = append(differencePercentageS, plotter.XY{
      X: degree64,
      Y: percentageDifference64,
    })

    i++
  }

  return differencePercentageS
}

//getDifferencePercentage returns difference percentage between simple and actual pendulum's period
func getDifferencePercentage(gravity, length, periodActual *big.Float) *big.Float {
  //simple period = 2π√(length/gravity)
  periodSimple := bigfloatCreate(0).Quo(length, gravity)
  periodSimple.Sqrt(periodSimple)
  //Multiply separately to keep pi's high precision
  periodSimple.Mul(periodSimple, bigfloatCreate(math.Pi))
```

```go
  periodSimple.Mul(periodSimple, bigfloatCreate(2))

  //period percentage difference
  percentageDifference := bigfloatCreate(0).Quo(periodActual, periodSimple)
  percentageDifference.Sub(percentageDifference, bigfloatCreate(1))
  percentageDifference.Mul(percentageDifference, bigfloatCreate(100))

  return percentageDifference
}

//periodGet returns the period based on parameters by numerical analysis using euler's algorithm
func periodGet(gravity, length, degreeStart, timeDelta, angularVelocity *big.Float) *big.Float {
  if degreeStart.Cmp(bigfloatCreate(0)) == 0 {
    return bigfloatCreate(math.Inf(1))
  }

  //constant = gravity / length
  constant := bigfloatCreate(0).Quo(gravity, length)
  constant.Neg(constant)

  //start values
  alphaStart := degreeToRadian(degreeStart)
  alpha := bigfloatCreate(0).Copy(alphaStart)

  i := 0
  for {
    //current time
    //calculate each time to get exact value
    time := bigfloatCreate(0).Mul(bigfloatCreate(float64(i)), timeDelta)

    //period check
    if alpha.Cmp(alphaStart) != -1 && time.Cmp(bigfloatCreate(0)) != 0 {
      return time
    }

    //f'' calc
    alpha64, _ := alpha.Float64()
    angularAcceleration := bigfloatCreate(0).Mul(constant, bigfloatCreate(math.Sin(alpha64)))

    //f' = f' + f''dt
    angularVelocityDelta := bigfloatCreate(0).Mul(angularAcceleration, timeDelta)
    angularVelocity.Add(angularVelocity, angularVelocityDelta)

    //f = f + f'dt
    alphaDelta := bigfloatCreate(0).Mul(angularVelocity, timeDelta)
    alpha.Add(alpha, alphaDelta)

    i++
  }
}
```