Pergerakan Flagella dengan Gaya Pegas

Aufa Nu'man Fadhilah Rudiawan | aufanu39man@gmail.com
Florentin Anggraini Purnama | florenpan@gmail.com
Muhammad Fauzan Girindra | fauzangir@hotmail.com
Jamaludin Muchtar | jamalmuchtar7@gmail.com

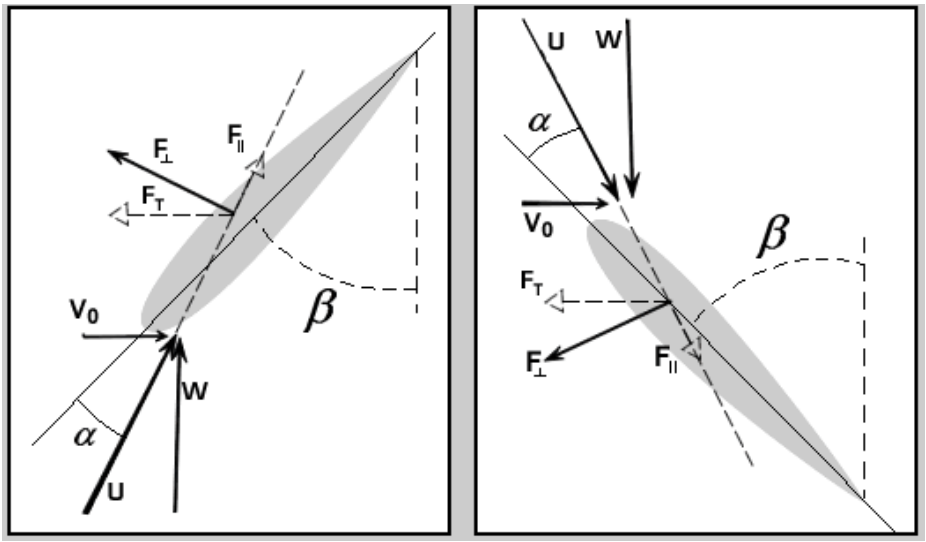
Mensimulasikan pergerakan flagella dengan menggunakan model grains yang dihubungkan dengan sebuah pegas sebagai ekor flagella. Variasi parameter dilakukan untuk kecepatan *U*, densitas fluida ρ, konstanta gaya drag (KDRAG), konstanta gaya lift (KL), konstanta pegas (KS), dan frekuensi dari kecepatan. Pergerakan flagella dipengaruhi oleh variabel kecepatan *U* dan rasio antara konstanta gaya drag (KDRAG), konstanta gaya lift (KL),

Pendahuluan

Mempelajari pergerakan ikan membuat peneliti dapat mendesain kendaraan dan robot yang dapat menavigasi lingkungan perairan dengan lihai. Untuk mempelajari hal tersebut diperlukan analisis interaksi struktur fluida ikan dan lingkungannya. Selama beberapa juta tahun, fenomena evolusi membuat ikan dapat bergerak dengan cepat dan mudah melewati lingkungan perairan yang mereka sebut 'rumah'. Lingkungan seperti itu, memberikan tantangan bagi kendaraan dan robot buatan manusia, sebagaimana perairan yang kelam kabut dan kurang cahaya menyulitkan mereka dalam navigasi. Untuk mengatasi masalah ini, seringkali ikan digunakan sebagai sumber inspirasi bagi perkembangan desain robot air.[1] Akan tetapi, dalam artikel ini, kelompok kami menggunakan pergerakan flagella untuk dipelajari, dengan menggunakan beberapa parameter acuan seperti frekuensi dan amplitudo.

Teori

Seperti dilansir pada referensi untuk kasus pergerakan thunniform[2], kami menggunakan penjelasan yang serupa untuk menjelaskan mekanisme pergerakan dari flagella. Ekor dari flagella dapat kita gambarkan sebagai butiran-butiran (granular) yang berosilasi. Dari sisi samping, ekor tersebut berosilasi ke atas dan ke bawah. Diagram bebas dari sisi pandang ini ditunjukkan dalam gambar berikut.



Gambar 1. *Diagram bebas ketika butiran berosilasi ke bawah (kiri) dan ke atas (kanan)*

Skema pada gambar kiri mengilustrasikan ketika butiran bergerak ke bawah dengan gaya W dan secara keseluruhan (flagella) bergerak ke depan dengan kecepatan V_0. Kemudian laju yang tampak merupakan penjumlahan dari kedua vektor V_0 dan W. Laju U dihasilkan dari gaya 'angkat' yang tegak lurus terhadap U.

Untuk menambah daya dorong, dapat dilakukan dengan memperbesar W, yang bertujuan untuk memperbesar U dan α. Untuk butiran flagella yang berosilasi, W dapat diperbesar dengan cara menambah frekuensi osilasi f atau amplitudo osilasi H. Sebagai contoh, untuk nilai frekuensi tetap, ketika nilai H bertambah maka nilai W juga bertambah karena butiran harus bergerak lebih banyak dalam waktu yang sama. Demikian pula, ketika memperbesar f dalam H tetap, maka nilai W akan bertambah karena butiran tersebut kekurangan waktu untuk menjangkau jarak yang sama. Maka, menambah frekuensi dan/atau amplitude dapat menambah daya dorong.

Efisensi daya dorong diperoleh dengan menggunakan rumusan berikut, dengan rata-rata berkisar 0.3.

$$\frac{fH}{V_0} \approx 0.30$$

Pertama-tama, 50 grain (termasuk kepala flagellum) digambar. Metode dinamika grain yang digunakan dalam makalah ini menerapkan 3 hukum gaya, yaitu gaya lift,

$$F_{Li} = \frac{1}{2} \times c_L \times \rho \times (v_i)^2 \times A$$

gaya drag,

$$F_{Di} = \frac{1}{2} \times c_D \times \rho \times (v_i)^2 \times A$$

dan gaya pegas

$$F_S = -\frac{1}{2} k\, \Delta x$$

dengan v adalah kecepatan grain relatif terhadap aliran fluida, A adalah luas penampang melintang dari grain, ρ adalah densitas fluida, c_L adalah konstanta gaya lift, c_D adalah konstanta gaya drag, k adalah konstanta gaya pegas, dan Δx adalah jarak antara suatu grain dan grain yang terletak di depannya.

Karena gerakan flagellum sangat kompleks dan sangat bervariasi, maka digunakan pergerakan ikan untuk mendekati pergerakan flagellum, meskipun pada perhitungannya tidak begitu persis. Pusat pergerakkan seluruh flagellum bergantung pada grain ke-2 dari kepala yang ditandai dengan warna merah.

Untuk sudut pergerakan flagellum, digunakan $2\pi ft$ agar gerakannya melingkar bervariasi secara smooth terhadap waktu.

Metode numerik

Untuk menggambarkan posisi setiap grain pada setiap waktu, digunakan metode algoritma Euler yang menjumlahkan semua gaya

$$\sum F_i = ma_i$$

$$v_i(t + \Delta t) = v_i(t) + a_i(t)\Delta t \quad (1.13)$$

$$x_i(t + \Delta t) = x_i(t) + v_i(t)\Delta t \quad (1.14)$$

Algoritma

Simulasi pergerakan flagella dengan model butiran-butiran yang berosilasi dilakukan dengan menggunakan algoritma berikut ini:

L1.	*U, freq, kS, kDrag, ρ, kL ?*
L2.	*N=50, dt=1E-3*
L3.	*M=0.1, D=0.2, x0=-13.3, x_i=x0+i*D, y_i=-7, v=0*
L4.	*Ux = U*cos(2*PI*freq*t)*
L5.	*Uy = U*sin(2*PI*freq*t)*
L6.	*V[N-2] = Ux + Uy*

L7.	*A= PI*(D*0.5)*(D*0.5)*
L8.	*SF[i] = (0.5*kL*rho*A*Ux*Ux)+(0.5*kL*rho*A*Uy*Uy,), i:0~N*
L9.	*SF[i] = SF[i]+ (0.5*kD*rho*A*Ux*Ux)+(0.5*kD*rho*A*Uy*Uy,), i:0~N*
L10.	$SF[i] = SF[i]+(-kS*((r_i-r0_i)-(r_{i-1}-r0_{i-1}))) \; i:N$
L11.	$SF[i] = SF[i]+(-kS*((r_i-r0_i)-(r_{i+1}-r0_{i+1}))) \; i:0$
L12.	$SF[i] = SF[i]+(-kS*((r_i-r0_i)-(r_{i+1}-r0_{i+1})))+(-kS*((r_i-r0_i)-(r_{i-1}-r0_{i-1}))) \; i:1\sim N-1$
L13.	$a_i=SF[i]/m, \; i:0\sim N$
L14.	$V_i=v_i+a_i*dt$
L15.	$r_i=r_i+v_i*dt$

Hasil dan diskusi

Variasi kecepatan sesuai dengan fisis, yaitu ketika U dinolkan, flagellum tidak berpindah tempat. Ketika dibesarkan, flagellum bergerak ke arah kanan atas dengan cepat dan ketika dikecilkan, flagellum bergerak ke arah kanan atas dengan kecepatan yang lebih lambat. Flagellum terus bergerak ke arah kanan atas karena komponen y dari gerak flagellum yang lebih besar dari komponen x dari gerak flagellum.

Ketika densitas fluida (**ρ**) dikecilkan, gerak flagellum semakin lambat, sesuatu yang bertentangan dengan fakta fisis. Seharusnya ketika densitas fluida semakin kecil, semakin sulit flagellum bergerak dalam fluida. Hal ini mungkin karena semakin kecil fluida, semakin kecil contact force yang didapatkan dari fluida untuk gaya lift flagellum sehingga gerakannya pun juga menjadi semakin lambat.

Ketika konstanta gaya drag (KDRAG) dan konstanta gaya lift (KL) disamakan, flagellum tidak bergerak. Hal ini sesuai dengan kenyataan fisisnya, yang artinya flagellum sedang berada dalam keadaan cruise. Ketika KL lebih besar dari KDRAG, flagellum dapat bergerak ke arah kanan atas. Hal ini juga sesuai dengan kenyataan fisisnya. Ketika KDRAG lebih besar dari KL, flagellum malah bergerak ke arah kiri bawah. Hal ini karena belum dilakukan pengaturan bahwa gaya drag tidak bisa lebih besar dari gaya lift.

Ketika konstata gaya pegas (KS) diperbesar, struktur flagellum hancur dan kode pun menjadi error. Ketika KS diperkecil, grain dari flagellum menjadi tercecer dan gerakannya menjadi tidak sinkron. Hal ini karena ketika KS kecil, artinya sedikit saja gaya akan menghasilkan perubahan posisi yang besar terhadap grain.

Ketika frekuensi dinaikkan, gerak flagellum semakin lama menjadi ke arah horizontal. Ketika frekuensi diturunkan, gerak flagellum malah melayang ke atas. Hal ini mungkin karena kesalahan metode numerik.

Kesimpulan

Hasil simulasi gerak flagellum sudah cukup mendekati keadaan sebenarnya meskipun masih terdapat kekurangan dari metode numerik.

Referensi

1. "Undulatory Swimming", Southern Methodist University Lyle School of Engineering, URL lyle.smu.edu/propulsion/Pages/undulatory.htm [20170521].
2. Fairclough, Caty, "Studying the Swimming Patterns of Fish with Simulation", Blogs, January, 4th 2016, URL https://www.comsol.com/blogs/studying-the-swimming-patterns-of-fish-with-simulation/ [20170521].
3. https://www.grc.nasa.gov/www/K-12/airplane/ldrat.html [20170522]
4. https://www.grc.nasa.gov/www/K-12/airplane/drag1.html [20170522]

Lampiran

A. Keterangan penggunaan program dan skrip

Tabel 1.2 Program dan skrip serta hasil keluarannya.

Program / Skrip	Piranti lunak	Keluaran	Lampiran
index4a.html	HTML	v_f	B
draw2da.js	Javascript	Gambar 1.2	C
forcea.js	Javascript	Gambar 1.3	D
graina.js	Javascript	Gambar 1.4	E
layouta.js	Javascript	Gambar 1.5	F
mydyanmicsa.js	Javascript	Gambar 1.6	G
vect3a.js	Javascript		H

B. Skrip program index4a.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="ISO-8859-1">
		<title>vect3</title>
		<script src="Grains4/vect3a.js"></script>
		<script src="Grains4/graina.js"></script>
		<script src="Grains4/forcea.js"></script>
```

```
            <script src="Grains4/mdynamicsa.js"></script>
            <script src="Grains4/draw2da.js"></script>
            <script src="Grains4/layouta.js"></script>
      </head>
      <body>
            <script>

                  var timer1;
                  var N = 50;
                  var grains = new Array(N);
                  var kL; //lift constant
                  var magU; //flow speed
                  var theta; //angle

                  layout();
                  initiate();

                  Mdynamics.setdt(1E-3); //set timestep

                  //initiate all positions and parameters
of the head and grains
                  function initiate() {

                        m1 = 0.1; //mass of all grains
                        c1 = "#000"; //border color of all
grains
                        D = 0.2; //diameter of the tails
                        x0 = -13.3; //initial position
                        var i = 0;
                        for(var ix = 0; ix <= N; ix++) {
                              if  (ix==N){  //this  is  for
head
                                    grains[i] = new Grain;
                                    grains[i].i = i;
                                    x = x0 + ix*D + 0.12;
//head position
                                    grains[i].r    =    new
Vect3(x, -7, 0);
                                    grains[i].r0   =   new
Vect3(x, -7, 0); //initial head position
                                    grains[i].m = m1;
                                    grains[i].c = c1;
                                    grains[i].D = D + 0.4;
//diameter of head
                              }else{      //this   is   for
tail grains
```

```
                                    grains[i] = new Grain;
                                    grains[i].i = i;
                                    x = x0 + ix*D; //tail
grain position
                                    grains[i].r   =   new
Vect3(x, -7, 0);
                                    grains[i].r0   =   new
Vect3(x, -7, 0); //initial tail grain position
                                    grains[i].m = m1;
                                    grains[i].c = c1;
                                    grains[i].D = D;
                              }
                              i++;
                        }

                        //plot everything initially
                        for(var iN = 0; iN <= N; iN++) {
                              plotParticle(grains[iN]);
                        }
                  }

                  iter = 0;

                  function run() {
                        grains[N-2].c = "#f50";//make the
locomotor grain red

                        getParams();//get parameters like
freq, ampl and so on

                        /*let's input U and angle
                        if(iter%2 == 0)
                        {
                              theta  =  -Math.PI/3;  //tail
goes down
                        }else if(iter%2 != 0)
                        {
                              theta  =  Math.PI/3;  //tail
goes up
                        }*/

                        Ux                                  =
magU*Math.cos(2*Math.PI*freq*t); //horizontal of U
                        Uy                                  =
magU*Math.sin(2*Math.PI*freq*t); //vertical of U
```

```
                        vecU    =    new    Vect3(Ux,Uy,0);
//vector components of flow speed

                        //pake grain N-2 sbg locomotor krn
nanti kepalanya putus dr tail
                        grains[N-2].v = vecU;

                        // Prepare variable for saving sum
of forces
                        SF = new Array(N);
                        for(var iN = 0; iN <= N; iN++) {
                              SF[iN] = new Vect3;
                        }

                        //calculate lift force
                        for(var iN = 0; iN <= N; iN++)
                        {
                              var           f           =
Force.lift(grains[iN]);
                              SF[iN]  =  Vect3.add(SF[iN],
f);
                        }

                        // Calculate drag force
                        for(var iN = 0; iN <= N; iN++) {
                              var           f           =
Force.drag(grains[iN]);
                              SF[iN]  =  Vect3.add(SF[iN],
f);
                        }

                        // Calculate spring force
                        for(var iN = N; iN >= 0; iN--) {
                              var f = new Vect3;
                              if (iN==(N)){
                                    f                   =
Force.spring(grains[iN], grains[iN-1]);
                              }else if (iN==0){
                                    f                   =
Force.spring(grains[iN], grains[iN+1]);
                              }else{
                                    f                   =
Vect3.add(Force.spring(grains[iN],
grains[iN+1]),Force.spring(grains[iN], grains[iN-1]));
                              }
```

```
                              SF[iN]  =  Vect3.add(SF[iN],
f);
                        }

                        var             hout            =
document.getElementById("hout");
                        hout.innerHTML =  grains[N-1].r.y;
//show the position
                        var str = "\nt = " + t + " s";
//put current time in textarea
                        hout.innerHTML += str;

                        clearCurrentFigure();
                        for(var iN = 0; iN <= N; iN++) {
                              //this one is simply to draw
the grains and head
                              plotParticle(grains[iN]);
                        }

                        //make it come again from left,
infinite loop
                        if(grains[N-1].r.x >= 8)
                        {
                              initiate();
                        }

                        for(var iN = 0; iN <= N; iN++) {
                              Mdynamics.Euler(SF[iN],
grains[iN]);
                        }
                        Mdynamics.inct(); //increase time
void

                        iter++;
                  }
            </script>
      </body>
</html>
```

C. Skrip program draw2da.js

```
// Define global variables for real world coordinates
var xwmin = 0;
var ywmin = 0;
var xwmax = 0;
var ywmax = 0;
```

```
// Define global variables for canvas coordinate
var xcmin = 0;
var ycmin = 0;
var xcmax = 0;
var ycmax = 0;

// Define current canvas
var currentFigure = "";
var figureBackground = "#fff";

// Set real world coordinates
function setWorldCoordinates(xmin, ymin, xmax, ymax) {
      xwmin = xmin;
      ywmin = ymin;
      xwmax = xmax;
      ywmax = ymax;
}

// Set canvas coordinates
function setCanvasCoordinates(canvasId) {
      currentFigure = canvasId;
      var c = document.getElementById(canvasId);
      xcmin = 0;
      ycmin = c.height;
      xcmax = c.width;
      ycmax = 0;
}

// Transform x
function transX(x) {
      var xx = (x - xwmin) / (xwmax - xwmin) * (xcmax -
xcmin);
      xx += xcmin;
      var X = parseInt(xx);
      return X;
}

// Transform y
function transY(y) {
      var yy = (y - ywmin) / (ywmax - ywmin) * (ycmax -
ycmin);
      yy += ycmin;
      var Y = parseInt(yy);
      return Y;
}
```

```
// transform angle
// always put true in plots because real angles are drawn
anticlockwise
var rangle;
var vangle;
function transAngle(rangle)
{
      if(rangle == 0)
      {
            vangle = 0;
      }else if(rangle == 1 || rangle == -1)
      {
            vangle = 1;
      }else if(rangle == 2 || rangle == -2)
      {
            vangle = 2;
      }
      else if(rangle > 0 && rangle < 1) // lower half
positive
      {
            vangle = 2-rangle;
      }else if(rangle > 1 && rangle < 2) // upper half
positive
      {
            vangle = 2-rangle;
      }else if(rangle < 0 && rangle > -1) // lower half
negative
      {
            vangle = -rangle;
      }else if(rangle < -1 && rangle > -2) // upper half
negative
      {
            vangle = -rangle;
      }
      return vangle;
}

// Clear current figure
function clearCurrentFigure() {
      var c = document.getElementById(currentFigure);
      var ctx = c.getContext("2d");
      ctx.fillStyle = figureBackground;
      ctx.fillRect(xcmin, ycmax, xcmax, ycmin);
}

// Plot particle
```

```
function plotParticle(p) {
      var x = p.r.x;
      var y = p.r.y;
      var D = p.D;

      var xx = transX(x);
      var yy = transY(y);
      var DD = transX(D) - transX(0);

      var c = document.getElementById(currentFigure);
      var ctx = c.getContext("2d");
      ctx.strokeStyle = p.c;
      ctx.lineWidth = 1.5;
      ctx.beginPath();
      ctx.arc(xx, yy, 0.5 * DD, 0, 2 * Math.PI);
      ctx.stroke();

      /*
      ctx.font = "10px Times New Roman"
      ctx.fillStyle = "black";
      ctx.textAlign = "center";
      ctx.textBaseline="middle";
      ctx.fillText(p.i, xx, yy);
      */

}
```

D. Skrip program forcea.js

```
// Class of Force
function Force() {

}

//spring force is physically correct already
// Define Spring force
Force.spring = function(p1, p2) {
      var m1 = p1.m;
      var m2 = p2.m;
      var r1 = p1.r;
      var r2 = p2.r;
      var r10 = p1.r0;
      var r20 = p2.r0;
      var  r  =  Vect3.sub(Vect3.sub(r1,  r10),Vect3.sub(r2,
r20));
      var u = Vect3.uni(r);
      var f = Vect3.mul(-kS,r);
```

```
      return f;
}

//should be parallel to locomotor movement... 1D assumption
totally wrong
//define drag force
Force.drag = function(p) {
      var v = p.v;
      var D = p.D;
      var A = Math.PI*(D*0.5)*(D*0.5);
      v2 = new Vect3 (-v.x*v.x,v.y*v.y,0);
      var   f   =   new   Vect3(-0.5*kD*rho*A*Uy*Uy,   -
0.5*kD*rho*A*Ux*Ux, 0);
      return f;
}

//define lift force
Force.lift = function(p)
{
      var v = p.v;
      var D = p.D;
      var A = Math.PI*(D*0.5)*(D*0.5);
      var    f    =    new    Vect3(0.5*kL*rho*A*Uy*Uy,
0.5*kL*rho*A*Ux*Ux, 0);
      return f;
}
```

E. Skrip program graina.js

```
// Class of Grain
function Grain() {
      if(arguments.length == 6) {
            this.i = arguments[0];
            this.m = arguments[1];
            this.D = arguments[2];
            this.q = arguments[3];
            this.c = arguments[4];
            this.r = arguments[5];
            this.v = arguments[6];
      } else if(arguments.length == 1) {
            this.i = arguments[0].i;
            this.m = arguments[0].m;
            this.D = arguments[0].D;
            this.q = arguments[0].q;
            this.c = arguments[0].c;
            this.r = arguments[0].r;
```

```
            this.v = arguments[0].v;
            //still don't get this, probably the input is
an array or
            //function or class?
      } else {
            this.i = 0;
            this.m = 1.0;
            this.D = 1.0;
            this.q = 1.0;
            this.c = "#000";
            this.r = new Vect3;
            this.r0 = new Vect3;
            this.v = new Vect3;
      }
      this.strval = function() { //this is to return the
components
            var str = "("
            str += this.i + ", ";
            str += this.m + ", ";
            str += this.D + ", ";
            str += this.q + ", ";
            str += this.c + ", ";
            str += this.r.strval() + ", ";
            str += this.v.strval() + ")";
            return str;
      }
}
```

F. Skrip program layouta.js

```
function layout() {
      // Create left division
      var ldiv = document.createElement("div");
      document.body.appendChild(ldiv);
      ldiv.style.border = "0px black solid";
      ldiv.style.height = "402px";
      ldiv.style.float = "left";
      ldiv.style.width = "252px";

      // Create right division
      var rdiv = document.createElement("div");
      document.body.appendChild(rdiv);
      rdiv.style.border = "0px black solid";
      rdiv.style.height = "402px";
      rdiv.style.float = "left";

      // Create text area for showing time
```

```
	var ta1 = document.createElement("textarea");
	ldiv.appendChild(ta1);
	ta1.style.color = "black";
	ta1.style.background = "white";
	ta1.rows = "2";
	ta1.cols = "32";
	ta1.style.overflowY = "scroll";
	ta1.style.display = "block";
	ta1.id = "hout";

	//text area for input
	var ta2 = document.createElement("textarea");
	ldiv.appendChild(ta2);
	ta2.id = "ta2";
	ta2.style.color = "black";
	ta2.style.background = "white";
	ta2.rows = "9";
	ta2.cols = "32";
	ta2.style.overflowY = "scroll";
	ta2.style.display = "block";
	ta2.value = defaultParams();

	// Create canvas for drawing
	var c = document.createElement("canvas");
	rdiv.appendChild(c);
	c.id = "canvas";
	c.style.border = "1px solid #999";
	c.style.background = "white";
	c.width = "800";
	c.height = "400";
	//c.style.float = "left";
	var ctx = c.getContext("2d");

	// Prepare canvas
	setCanvasCoordinates("canvas");
	setWorldCoordinates(-8, -8, 8, 8);

	// Create start button
	var b1 = document.createElement("button");
	ldiv.appendChild(b1);
	b1.innerHTML = "Start";
	b1.onclick = function() {
		if(b1.innerHTML == "Start") {
			b1.innerHTML = "Stop";
			timer1 = setInterval(run, 1); //run the
function 'run'
```

```
            } else {
                  b1.innerHTML = "Start";
                  clearInterval(timer1);
            }
      }

      // Create default Button
      var b2 = document.createElement("button");
      ldiv.appendChild(b2);
      b2.innerHTML = "Default";
      b2.onclick = function() {
            var ta = document.getElementById("ta2");
            ta.value = defaultParams();
      }
}

// Generate default parameters
function defaultParams() {
      var params = "";
      params += "U\t 50\n";
      params += "FREQ\t 30.0\n";
      params += "KS\t 5E4\n";
      params += "KDRAG\t 0.47\n";
      params += "RHO\t 50\n";
      params += "KL\t 0.50\n";

      return params;
}

// Get parameters from textarea nText
function getParams()
{
      var ta = document.getElementById("ta2");
      var lines = ta.value.split("\n"); //idk what these're
for
      var Nlines = lines.length;
      /*if (lines[Nlines - 1].length == 0) {
            Nlines--;
      }*/
      //you  can  change  the  parameters  with  these  in
textarea
      for(var l = 0; l < Nlines; l++) {
            var cols = lines[l].split(" ");
            //var Ncols = cols.length;
            //still don't get what above is for
            if(cols[0] == "U\t")
```

```
            {
                  magU = parseFloat(cols[1]);
            }
            if(cols[0] == "FREQ\t") {
                  freq  =  parseFloat(cols[1]);  //convert
string to float
            }
            if(cols[0] == "AMPL\t") {
                  Amp = parseFloat(cols[1]);
            }
            if(cols[0] == "KS\t") {
                  kS = parseFloat(cols[1]);
            }
            if(cols[0] == "KDRAG\t") {
                  kD = parseFloat(cols[1]);
            }
            if(cols[0] == "RHO\t") {
                  rho = parseFloat(cols[1]);
            }
            if(cols[0] == "KL\t")
            {
                  kL = parseFloat(cols[1]);
            }
      }

}
```

G. Skrip program mdynamicsa.js

```
// Define some global constants
dt = 0.01;
t = 0;

// Class of Mdynamics
function Mdynamics() {
      //this function can hold many things, depends on how
many
      //arguments it can have, it's just a different way
compared
      //to grain.js
}

// Set time step and reset time
Mdynamics.setdt = function(dtt) {
      dt = dtt;
      t = 0;
}
```

```
// Perform Euler integration
Mdynamics.Euler = function(SF, p) {
      var m = p.m;
      var r = p.r;
      var v = p.v;
      var a = Vect3.div(SF, m); //get acceleration
      v = Vect3.add(v, Vect3.mul(a, dt));//add it with
current speed
      r = Vect3.add(r, Vect3.mul(v, dt));
      p.r = r;//make it current grain position
      p.v = v;//make it current grain velocity
}

// Increase time
Mdynamics.inct = function() {
      t += dt;
}
```

H. Skrip program vect3.js

```
// Class of Vect3
function Vect3() {
      // Define some constructor types
      if(arguments.length == 3) {
            this.x = arguments[0];
            this.y = arguments[1];
            this.z = arguments[2];
      } else if(arguments.length == 1){
            this.x = arguments[0].x;
            this.y = arguments[0].y;
            this.z = arguments[0].z;
      } else {
            this.x = 0;
            this.y = 0;
            this.z = 0;
      }
      this.strval = function() { //to return the vector
components
            var str = "(";
            str += this.x + ", ";
            str += this.y + ", ";
            str += this.z + ")";
            //document.write(str);
            return str;
      }
```

```
}

// Define addition of Vect3
Vect3.add = function(r1, r2) {
      var r = new Vect3;
      r.x = r1.x + r2.x;
      r.y = r1.y + r2.y;
      r.z = r1.z + r2.z;
      return r;
}

// Define substraction of Vect3
Vect3.sub = function(r1, r2) {
      var r = new Vect3;
      r.x = r1.x - r2.x;
      r.y = r1.y - r2.y;
      r.z = r1.z - r2.z;
      return r;
}

// Define dot product of Vect3
Vect3.dot = function(r1, r2) {
      var l = r1.x * r2.x;
      l += r1.y * r2.y;
      l += r1.z * r2.z;
      return l;
}

// Define cross product of Vect3
Vect3.crs = function(r1, r2) {
      var r = new Vect3;
      r.x = r1.y * r2.z - r1.z * r2.y;
      r.y = r1.z * r2.x - r1.x * r2.z;
      r.z = r1.x * r2.y - r1.y * r2.x;
      return r;
}

// Define multiplication with scalar
Vect3.mul = function(a, b) {
      var r = new Vect3;
      if(a instanceof Vect3) {
            r.x = a.x * b;
            r.y = a.y * b;
            r.z = a.z * b;
      } else if(b instanceof Vect3) {
            r.x = a * b.x;
```

```
            r.y = a * b.y;
            r.z = a * b.z;
      }
      return r;
}

// Define division with scalar
Vect3.div = function(a, b) {
      var r = new Vect3;
      r.x = a.x / b;
      r.y = a.y / b;
      r.z = a.z / b;
      return r;
}

// Define length of Vect3
Vect3.len = function(r) {
      var l2 = Vect3.dot(r, r);
      var l = Math.sqrt(l2);
      return l;
}

// Define unit vector
Vect3.uni = function(a) {
      var l = Vect3.len(a);
      var r = Vect3.div(a, l);
      return r;
}

// Define negative of a vector
Vect3.neg = function(a) {
      var r = Vect3;
      r.x = -a.x;
      r.y = -a.y;
      r.z = -a.z;
      return r;
}
```