32 Runge-Kutta

O método de Runge-Kutta é um dos mais populares para realizar a integração numérica de equações diferenciais que não podem ser resolvidas analiticamente, como é o caso de sistemas de muitos corpos em mecânica. Detalhes técnicos sobre o método podem ser encontrados em qualquer bom livro de métodos numéricos (um clássico é o *Numerical Recipes*, de W. H. Press, S. A. Teukolsky, W. T. Vetterling e B. P. Flannery, da Cambridge University Press) e em um grande número de sites na Internet.

No exemplo a seguir, é utilizado o método de Runge-Kutta de 4a. ordem para calcular a trajetória de uma partícula em um campo de força central. No caso, são duas partículas carregadas com uma interação elétrica entre elas. O script essencialmente simula o experimento de espalhamento de partículas alfa por uma folha de ouro utilizado por Ernst Rutherford e colaboradores em 1911 para demonstrar a existência do núcleo atômico.

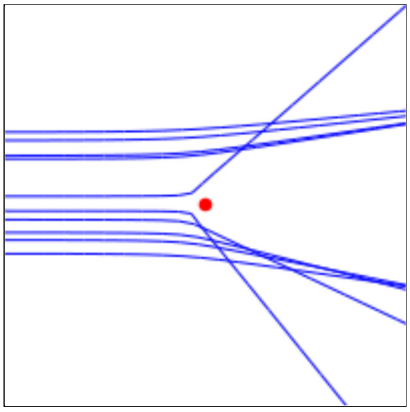
exemplo-32-1.html

```
<p>
<div align="center">
<canvas id='cnvRK4' style='border: 1px solid;'
           width='200' height='200'></canvas>
</div>
</p>

<script>

function fixCoords(lctx,cw,ch,xmin,xmax,ymin,ymax) {
   var m11 = cw/(xmax-xmin);
   var m12 = 0;
   var m21 = 0;
   var m22 = ch/(ymin-ymax);
   var dx  = - cw*xmin/(xmax-xmin);
   var dy  = - ch*ymax/(ymin-ymax);
   lctx.transform(m11,m12,m21,m22,dx,dy);
   var esp = Math.max(m11,m22);
   lctx.lineWidth = 1/esp;
}

function calcE(x,y) {
   var C = 9e9 * 79 * 1.6e-19;
   var den = Math.pow((x*x+y*y),3/2);
```

```
   var Ex = C * x/den;
   var Ey = C * y/den;
   return {x:Ex,y:Ey};
}

function passoRK4() {
   var kx1,ky1,kx2,ky2,kx3,ky3,kx4,ky4;
   var E;

   xi = x;
   yi = y;

   E = calcE(x,y);
   kx1 = h * (Za/ma) * E.x;
   ky1 = h * (Za/ma) * E.y;
   x = xi + (h/2) * (vx + kx1/2);
   y = yi + (h/2) * (vy + ky1/2);

   E = calcE(x,y);
   kx2 = h * (Za/ma) * E.x;
   ky2 = h * (Za/ma) * E.y;
   x = xi + (h/2) * (vx + kx2/2);
   y = yi + (h/2) * (vy + ky2/2);

   E = calcE(x,y);
   kx3 = h * (Za/ma) * E.x;
   ky3 = h * (Za/ma) * E.y;
   x = xi + (h/2) * (vx + kx3);
   y = yi + (h/2) * (vy + ky3);

   E = calcE(x,y);
   kx4 = h * (Za/ma) * E.x;
   ky4 = h * (Za/ma) * E.y;
   vx = vx + (1/6) * (kx1+2*(kx2+kx3)+kx4);
   vy = vy + (1/6) * (ky1+2*(ky2+ky3)+ky4);
   x = xi + h * (vx+(1/6)*(kx1+kx2+kx3));
   y = yi + h * (vy+(1/6)*(ky1+ky2+ky3));

   // desenha o trecho da trajetória
   ctx.beginPath();
   ctx.strokeStyle = "blue";
   ctx.moveTo(xant,yant);
   ctx.lineTo(x,y);
   ctx.stroke();
   xant = x;
   yant = y;

   // reinicia quando bate em alguma parede
   if ((x<xmin)||(x>xmax)) {
      initVarsRK4();
      ntrajs++;
   }

   // para o temporizador quando atinge 10 trajetórias
   if (ntrajs==10) clearInterval(intervaloRK4);
}

function initVarsRK4() {
   x = xmin;
   y = (ymax-ymin)/10 * (Math.random()*2-1);
   yini = y;
   xant = x;
   yant = y;
   vx = va;
```

```
    vy = 0;
}

// define algumas variáveis globais
var xmin = -3e-12;
var xmax = +3e-12;
var ymin = -3e-12;
var ymax = +3e-12;
var x,y,yini,vx,vy,xi,yi;
var ma = 4 * 1.67e-27;              // massa da alfa
var Za = 2 * 1.6e-19;               // carga da alfa
var Ea = 2 * 1e6 * 1.6e-19;         // energia da alfa
var va = Math.sqrt(2 * Ea / ma);    // velocidade da alfa
var h = ((xmax-xmin)/va)/50;        // passo do Runge-Kutta

// canvas e sistema de coordenadas
var ctx = document.getElementById("cnvRK4").getContext("2d");
fixCoords(ctx,200,200,xmin,xmax,ymin,ymax);

// alvo no centro do canvas
ctx.beginPath();
ctx.fillStyle = "red";
ctx.arc(0,0,xmax/30,0,2*Math.PI,true);
ctx.fill();

var ntrajs = 0;

// inicializa variáveis do RK4
initVarsRK4();

// ativa o temporizador
var intervaloRK4 = setInterval("passoRK4()",1);

</script>
```

Resultado:

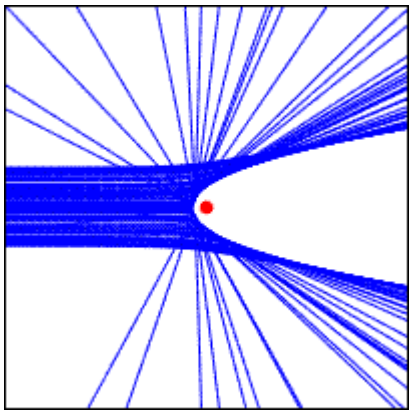


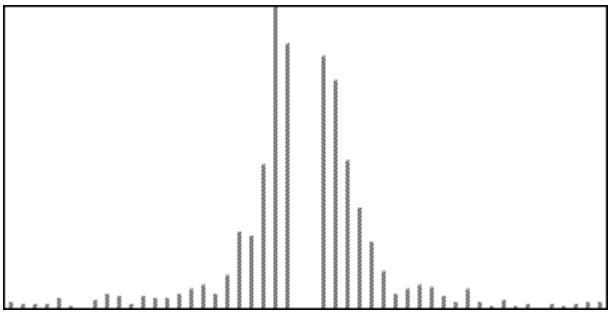
O método de Runge-Kutta propriamente dito está embutido na função `passoRK4()`. Note que, em termos computacionais, o método custa caro: são feitas quatro chamadas à função que calcula o campo elétrico para finalmente obter-se o valor do novo ponto, fazendo uma média ponderada das velocidades em quatro pontos intermediários.

Exercícios

1. Modifique o script para que gere também a distribuição para o ângulo de espalhamento das partículas, como na figura abaixo, que mostra a distribuição para 1000 partículas. Enquanto desenvolve sua solução, faça com que nem todas as trajetórias sejam desenhadas, pois o cálculo será muito lento. Uma possível estratégia para fazer isso é checar se o número de partículas já calculadas é um múltiplo de algum número e desenhar somente caso essa condição se verifique. No esboço de script a seguir, um laço `for` é utilizado para gerar as `nparts` desejadas e dentro dele o operador `%` (resto de divisão) é utilizado para fazer com que 1 trajetória seja desenhada a cada 10 cálculos.

```
for (var i=0;i<nparts;i++) {
   ...
   if ((i%10)==0) {
      ... código para desenhar trajetória ...
   }
   ...
}
```





−180 Ângulo de espalhamento (graus) +180

2. Generalize o script de modo que possam ser especificadas várias cargas fixas e que a trajetória calculada leve em conta os campos de todas elas. Você pode definir matrizes unidimensionais com as cargas e posições de cada uma delas e percorrê-la nos cálculos. Desenvolva um objeto `CARGA` que tenha como atributos a carga e as coordenadas e um método que retorne o valor do campo em um ponto dado.