Difusión de partículas en plasma
Víctor Martinell

Uno de los factores importantes a considerar en el estudio de los plasmas es la difusión de partículas debido a la turbulencia. Este determina la cantidad de plasma que se pierde al contenerlo confinado entre campos magnéticos.
Este factor suele ser complicado ya que, al ser causado por turbulencias, no hay una forma sencilla de representarlo.
En el artículo “Stochastic E x B particle transport, Robert G. Kleva and J. F. Drake” se prepone una primera aproximación sencilla de tomar en cuenta este factor de difusión.
Consideran un plasma en un campo magnético uniforme en dirección z

$$\mathbf{B} = B\hat{z}$$

Suponemos que hay una onda electrostática propagandose en dirección y con velocidad de fase V_{p1}.
El potencial de esta onda esta dado por:

$$\phi(x, y - V_{p1}t) = \phi_{01} cos(k_{1x}x) cos[k_{1y}(y - V_{p1}t)]$$

para una onda de baja frecuencia $\omega_1 = k_{1y} V_{p1} < \Omega$. Con k_{1y} el número de onda y Ω la frecuencia de giro.
La respuesta de una partícula del plasma a la onda esta dada por:

$$V_d = -c \bigtriangledown \phi \times \hat{z}/B$$

Entonces se pueden escribir las ecuaciones de movimiento de la partícula:

$$\frac{dx}{dt} = -\frac{c}{B}\frac{\partial \phi}{\partial y}$$

$$\frac{dy}{dt} = \frac{c}{B}\frac{\partial \phi}{\partial x}$$

Este par de ecuaciones fue el que se resolvió utilizando el método de integración en series de Taylor Basándose en el ejemplo de las ecuaciones de movimiento de Newton del paquete “TaylorSeries” de la libreria de julia. Se resolvió en primera instancia el sistema de ecuaciones utilizando el primer método. Este es mucho más lento pero la introducción de las ecuaciones es bastante directa:

```
function jetplasma{t<:real}( vec::array{taylor{t},1} )
   xt = vec[1]
   yt = vec[2]

   ord = xt.order
   for k = 0:ord-1
      knext = k+1
      # taylor expansions up to order k <--- this part makes it somewhat slower
      xtt = taylor( xt.coeffs[1:k+1], k)
      ytt = taylor( yt.coeffs[1:k+1], k)
      # eqs of motion <--- this is quite straight :-)
                                    auxt = ytt-vp*t
      xdot = -mag*dfiy(xtt, auxt)
      ydot = mag*dfix(xtt, auxt)
      #v = ( xdot^2 + ydot^2 )^(1/2)
      # the equations of motion define the recurrencies
      xt.coeffs[knext+1]  = xdot.coeffs[knext] / knext
      yt.coeffs[knext+1]  = ydot.coeffs[knext] / knext
   end
```

```
    return [ taylor(xt, ord), taylor(yt, ord)]
end
```

Donde las funciones “dfix” y “dfiy” fueron definidas previamente como las derivadas de la función ϕ con respecto a x y y respectivamente.
Con este método se logró reproducir satisfactoriamente la solución inicial que se mostraba en el artículo cuando no se tenía caos (es decir Vp = 0)

Vp0.pdf

Una vez visto que se iba por buen camino se comenzó a aumentar el término Vp para observar cómo cambiaban las soluciones. Se vio que con un valor Vp = 1 las se tenía un caos total. Pero como el método era muy lento era tardado y complicado graficar varios ejemplos para obtener más detalle. Entonces se implementó el segundo método que había en el ejemplo de Newton:

```
function jetPlasma2{T<:Real}( vec::Array{Taylor{T},1} )
    xT = vec[1]
    yT = vec[2]

    # Auxiliary quantities
    ord = xT.order
    sinkxx = zeros( T, ord+1 )#; sinkxx[1]=sin(xT[1])
    sinkyy = zeros( T, ord+1 )
    coskxx = zeros( T, ord+1 )
    coskyy = zeros( T, ord+1 )
    auxT = zeros( T, ord+1 )
    xDot = zeros( T, ord+1 )
    yDot = zeros( T, ord+1 )
    # Now the implementation
    for k = 0:ord-1
        knext = k+1
        # The right-hand size of the eqs of motion <--- this is adpated for this problem
        auxT[knext] = ky * (yT.coeffs[knext] - (vp*t.coeffs[knext]))
        sinkxx[knext], coskxx[knext] = TaylorSeries.sincosHomogCoef(k, kx * xT.coeffs, sinkxx, coskxx)
        sinkyy[knext], coskyy[knext]= TaylorSeries.sincosHomogCoef(k, auxT, sinkyy, coskyy)
        xDot[knext] = mag * fi0 * kx * TaylorSeries.mulHomogCoef(k, sinkxx, coskyy)
        yDot[knext] = -mag * fi0 * ky * TaylorSeries.mulHomogCoef(k, sinkyy, coskxx)
        # The equations of motion define the recurrencies
        xT.coeffs[knext+1]  = xDot[knext] / knext
        yT.coeffs[knext+1]  = yDot[knext] / knext
    end

                    #println(length(xT), length(yT))
                    #println(mean(xT)," ", mean(yT))

    return [ Taylor(xT, ord), Taylor(yT, ord) ]
end
```

Este método resulto ser muchísimo más rápido que el primero. Conseguía soluciones casi 100 veces más rápido que el primero.
Se intentó resolver el problema con más de 200 condiciones iniciales y sólo se llevó uno par de minutos. Pero por desgracia la cantidad de datos generada era de tal magnitud (del orden de 1 000 000) que el tiempo de graficación y en el que se requería para cargar las gráficas era enorme.
Por lo que se decidió usar sólo 25 condiciones iniciales para poder ver los resultados agilmente.
Así se lograron barrer varios intervalos de condicione iniciales y valores de Vp.
Se vio que el caos comenzaba a notares a partir de Vp > 0.002 Con un valor de 0.02 condiciones iniciales separadas por una distancia de 1 se traslapaban completamente.

Finalmente se modificó el programa para calcular la distancia entre 2 soluciones en el tiempo. Para esto se resolvieron las 2 soluciones en simultaneo. Como el “step size” es calculado dinámicamente el tiempo no coincidía en cada punto por lo que en cada momento se tomo para ambas soluciones el mínimo de los 2 “step size” calculados. Esto no debería afectar la solución de las ecuaciones hasta done entiendo de como funciona el método.
Así se obtuvieron gráficas de la separación de 2 soluciones. Específicamente se eligieron 2 que estaban casi en el límite de una circunferencia. -3.0, -3.0 y -2.8, -3.0
Cuando el valor de Vp estaba entre 0 y 0.03 las soluciones eran periódicas en el tiempo. A partir de 0.04 se puede considerar que comienza la difusión de las partículas ya que la solución se vuelve creciente.
Es posible cuantificar que tanto se separan estos puntos utilizando el coeficiente de Lyapunov. En teoría la velocidad con la que los puntos se separan describe una forma exponencial del tipo

$$|\delta \mathbf{Z}(t)| \approx e^{\lambda t} |\delta \mathbf{Z}_0|$$

Donde λ es el coeficiente de Lyapunov.
Pero al graficar las distancias para Vp entre 0.04 y 0.09 el crecimiento era caramente lineal. Y esto se confirmó cuando se graficó tiempo contra ln(distancia) y se vio como el crecimiento se tornaba logaritmico

dist-08.pdf es la distancia contra el tiempo a Vp = 0.08
logdist-08.pdf es la el logaritmo de la distancia contra el tiemp oa Vp = 0.08

Se intentaron tomar 2 puntos iniciales mucho más cercanos para buscar el comportamiento exponencial. Se consideraron a 0.05, 0.02 y 0.01 pero en ninguno de estos casos se oberva un crecimiento exponencial. Incluso con putnos separados 0.0001 sigue viendose un crecimiento lineal

Dist-01-06.pdf
Dist-0001-06.pdf

No logro ver alguna falla en el método o el programa. Con estos resultados se debe concluir que el factor Vp no genera caos al ser aumentaso, simplemente un desorden aparente. Eso parece poco probable pero por más que se mueven los parámetros el crecimiento sigue siendo lienal.
En el artículo se utiliza una segunda aproximación mas compleja para atacar el problema, aunque no se calcula el coeficiente de Lyapunov es evidente que el sisteme aes caótico.