Modelo de Tráfico

Temas Selectos de Física Computacional I

Aplicaciones Numéricas a Sistemas Dinámicos

Dr. Luis Benet Fernández

Gómez Bustamante Laura

Villegas Juárez Arturo

El modelo

1. El método General

Tenemos la ecuación diferencial:

$$\dot{x_i}(t) = \begin{cases} \frac{1}{K}\,(x_{i+1}(t) - x_i(t) - c_{i-1}) \\ v_{i_{max}} \end{cases}$$

Usemos el método de Taylor para integrar la ecuación anterior:

In [1]:
```
#Es necesario definir una variable tipo A : Auto
type Auto
    x
    vmax
    long
    function Auto(x,vmax,long)
        x = mod2pi(x)
        vmax = abs(vmax)
        long = long == zero(Float64) ? 2π/100.0 : abs(long)
        new(x,vmax,long)
    end
end
```

Modelo para 2 autos

1.1 El modelo para 2 autos

Tomemos el problema de dos autos en una pista circular de radio r. Las ecuaciones para este caso son:

$$\dot{x_1}(t) = \begin{cases} \frac{1}{K}\,(x_2(t) - x_1(t) - c_2) \\ v_{1_{max}} \end{cases}$$

y

$$\dot{x_2}(t) = \begin{cases} \frac{1}{K}\,(x_1(t) - x_2(t) - c_1) \\ v_{2_{max}} \end{cases}$$

Con c_1 y c_2 las dimensiones de los autos respectivamente

Como el movimiento es circular, tiene periodicidad. Entonces, el auto de hasta delante ve frente a él, al auto de hasta atrás; es decir el primero ve al último. Lo cual altera la ecuación de movimiento del primer auto.

In [13]:
```
using TaylorSeries
```

In [14]:
```
using PyPlot
```

```
INFO: Loading help data...
```

In [15]:
```
#Definimos la función dist(xi+1,xi), para i+1=j como: con j=modN(i+1), con N el número de autos
#dist(xj,xi) = xj>xi ? xj-xi : 2pi + xj- xi
function dist(x::Taylor, y::Taylor)
    r = x-y
    if r.coeffs[1] < zero(r.coeffs[1])
        r = 2pi + r
    end
    r
end

function dist(x, y)
    r = x-y
    if r < zero(x)
        r = 2pi + r
    end
    r
end
```

$$\dot{x_1}(t) = \begin{cases} \frac{1}{K}\,(dist[x_2(t) - x_1(t)] - c_2) \\ v_{1_{max}} \end{cases}$$

$$\dot{x_2}(t) = \begin{cases} \frac{1}{K}\,(dist[(x_1(t) - x_2(t)] - c_1) \\ v_{2_{max}} \end{cases}$$

In [5]:
```
#El radio de la pista es
r=50
#Tomamos la constante K=1
K=1;
```

In [6]:
```
#Las longitudes de los autos

ci=atan((1.5*rand(2)+3.5)/r);
```

In [7]:
```
C1=ci[1]
C2=ci[2];
```

In [8]:
```
#Las posiciones iniciales de los autos son xi0:

x10=mod2pi(0)
s=rand(1)
x20=x10+ci[2]+(2*π-abs(ci[1]))*s[1]
println("x1=", x10, "\t", "x2=", x20)
```

```
x1=0.0  x2=3.4990694801861264
```

In [9]:
```
#Las velocidades máximas de los autos son:

wi=((60*rand(2)+120)/r);
```

In [10]:
```
w1=Taylor(wi[1])
```

Out[10]:
```
Taylor{Float64}([3.52469],0)
```

In [11]:
```
w2=Taylor(wi[2])
```

Out[11]:
```
Taylor{Float64}([3.01741],0)
```

In [12]:
```
#Definimos vectores con la información de los Autos, del tipo: Auto(x,vmax,long)

Auto1=Auto(x10,wi[1],ci[1])
Auto2=Auto(x20,wi[2],ci[2])
println("Auto1=", Auto1, "\t", "Auto2=", Auto2 )
```

```
Auto1=Auto(0.0,3.5246870091829035,0.0723219139957003)
Auto2=Auto(3.4990694801861264,3.017408610560882,0.09075744970453546)
```

Para integrar

In [22]:
```
# Parámetros para el integrador de Taylor
const ordenTaylor = 18
const Epsilon = 1.0e-20;
```

Definimos el integrador de Taylor

In [14]:
```
function taylorStepper{T<:Real}( jetEqs::Function, vec0::Array{T,1}, order::Int64, Epsilon::T)
    n = length( vec0 )
    vec0T = [ Taylor([vec0[i]], order) for i=1:n ]
    vec1T = jetEqs( vec0T )

    # Step-size
    hh = Inf
    for i=1:n
        h1 = stepsize( vec1T[i], Epsilon )
        hh = min( hh, h1 )
    end
    hh = hh*0.0625

    # Valores a t0+h
    for i=1:n
        vec0[i] = mod2pi(evalTaylor(vec1T[i], hh ))
    end

    return hh, vec0
end
```

Out[14]:
```
taylorStepper (generic function with 1 method)
```

In [15]:
```
# Para el paso, de la Epsilon y los dos últimos coeficientes se tiene
function stepsize{T<:Real}(x::Taylor{T}, epsilon::Float64)
    ord = x.order
    h = Inf
    for k in [ord-1, ord]
        kinv = 1.0/k
        aux = abs( x.coeffs[k+1] )
        h = min(h, (epsilon/aux)^kinv)
    end
    return h
end
```

Out[15]:
```
stepsize (generic function with 1 method)
```

Las ecuaciones

In [16]:
```
function jetTraffic{T<:Real}( vec::Array{Taylor{T},1} )
    x1T = vec[1]
    x2T = vec[2]

    ord = x1T.order
    for k = 0:ord-1
        knext = k+1
        # Expansión de Taylor a orden k

        x1Tt = Taylor( x1T.coeffs[1:k+1], k)
        x2Tt = Taylor( x2T.coeffs[1:k+1], k)

        # Las ecuaciones de movimiento
        numerito = (dist(x2T.coeffs[1],x1T.coeffs[1])-C2) / K
        x1Dot = Taylor(0.0,k)
        if numerito <= wi[1]
            x1Dot = (dist(x2Tt,x1Tt)-C2) / K
        else
            x1Dot = Taylor(wi[1],k)
        end
        #@show x1Dot
        numerito = (dist(x1T.coeffs[1],x2T.coeffs[1])-C1) / K
        x2Dot = Taylor(0.0,k)
        if numerito <=  wi[2]
            x2Dot = (dist(x1Tt,x2Tt)-C1) / K
        else
            x2Dot = Taylor(wi[2],k)
        end
        #@show x2Dot



        # De las ecuaciones de movimiento se tienen las recurrencias
        x1T.coeffs[knext+1]  = x1Dot.coeffs[knext] / knext
        x2T.coeffs[knext+1]  = x2Dot.coeffs[knext] / knext

    end

    return [ Taylor(x1T, ord), Taylor(x2T, ord) ]
end
```

Out[16]:
```
jetTraffic (generic function with 1 method)
```

El integrador

In [17]:
```
function TrafficIntegration( time_max::Float64, jetEqs::Function, orden::Int64=ordenTaylor,
    epsilon::Float64=Epsilon )
    # Condiciones iniciales
    t0 = 0.0
    x10=mod2pi(0)
    s=rand(1)
    x20=x10+ci[2]+ 1.0e-14   #(2*π-abs(ci[1]))*s[1]


    # Vectores para graficar con PyPlot
    tV, x1V, x2V = Float64[], Float64[], Float64[]
    push!(tV, t0)
    push!(x1V, x10); push!(x2V, x20)

    # El ciclo principal con el step-size mínimo
    dt = 1.0
    while t0 < time_max && dt>1.0e-8
        # Aquí la integración
        dt, (x11, x21) = taylorStepper( jetEqs, [x10, x20], orden, Epsilon );
        t0 += dt
        push!(tV,t0)
        push!(x1V,x11); push!(x2V,x21)
        x10, x20 = x11, x21
    end

    return tV, x1V, x2V
end
```

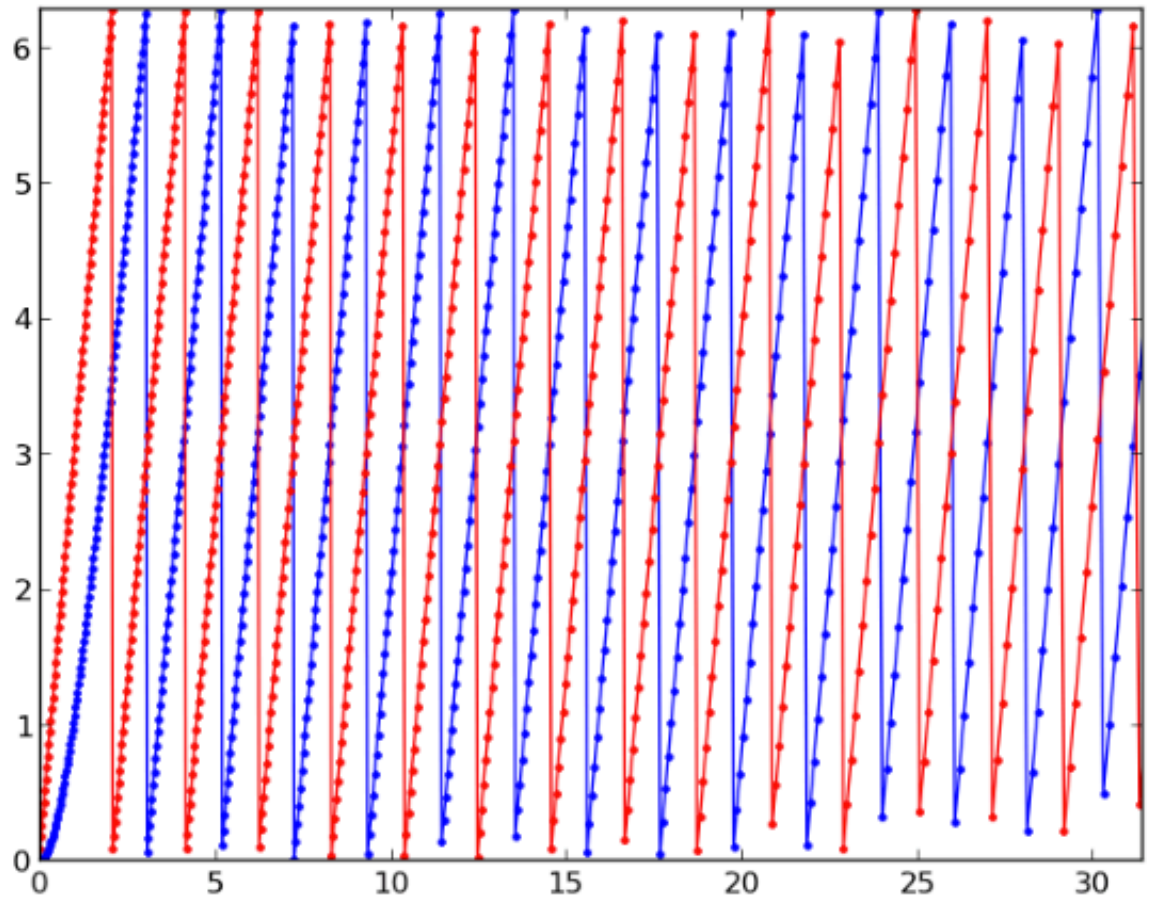
Out[17]:
```
TrafficIntegration (generic function with 3 methods)
```

In [18]:
```
#jetTraffic
tV1, x1V1, x2V1 = TrafficIntegration( 2pi, jetTraffic);
@time tV1, x1V1, x2V1 =
TrafficIntegration(  1000*2pi, jetTraffic);
```

```
elapsed time: 0.340686086 seconds (81241952 bytes allocated, 55.08% gc time)
```

Graficando

```
In [19]: plot(tV1,x1V1,"b.-", tV1, x2V1, "r.-")
         axis([0,10pi,0,2pi])
```




```
Out[19]: 4-element Array{Float64,1}:
```

Ahora….para N Autos

In [6]:
```
function condiciones(N::Int)
    #El radio de la pista es:
    R=50
    #Las longitudes de los N autos, en un solo vector
    C=atan((1.5*rand(N)+3.5)/R)


    #Las posiciones iniciales de los N Autos, en un solo vector
    X=Float64[]
        push!(X,0.0)
        for i=2:N
            Xi=X[i-1] + C[i] + 1.0e-14
            push!(X,Xi)
        end
    X = X -X[end]

    #Las velocidades máximas de los N Autos, en un solo vector
    W=((60*rand(N)+120)/R)

    Coches=Auto[]

    for i=1:N
        push!(Coches, Auto(X[i],W[i],C[i]))

    end

    return Coches

end
```

Out[6]:
```
condiciones (generic function with 1 method)
```

In [7]:
```
coches=condiciones(2)
```

Out[7]:
```
2-element Array{Auto,1}:
 Auto(-0.07108269808319219,2.8053846491591643,0.07430918110907939)
 Auto(0.0,3.4105608374763974,0.07108269808318218)
```

In [8]:
```
coches[1]
```

Out[8]:
```
Auto(-0.07108269808319219,2.8053846491591643,0.07430918110907939)
```

In [9]:
```
coches[2].w
```

Out[9]:
```
3.4105608374763974
```

Para el integrador

In [11]:
```
#Definimos el integrador de Taylor
function taylorStepper( jetEqs::Function, vec0::Array{Auto,1}, order::Int64, Epsilon::Float64)
    n = length( vec0 )
    vec1T = jetEqs( vec0 )

    # Step-size
    hh = Inf
    for i=1:n
        h1 = stepsize( vec1T[i], Epsilon )
        hh = min( hh, h1 )
    end
    #hh = hh*0.0625

    # Valores a t0+h
    for i=1:n
        aux = mod2pi(evalTaylor(vec1T[i], hh ))
        vec0[i].x = aux
    end

    return hh, vec0
end
```

Out[11]:
```
taylorStepper (generic function with 1 method)
```

In [12]:
```
# Para el paso, de la Epsilon y los dos últimos coeficientes se tiene
function stepsize{T<:Real}(x::Taylor{T}, epsilon::Float64)
    ord = x.order
    h = Inf
    for k in [ord-1, ord]
        kinv = 1.0/k
        aux = abs( x.coeffs[k+1] )
        h = min(h, (epsilon/aux)^kinv)
    end
    return h
end
```

Las ecuaciones

In [13]:
```
function jetTraffic(coches::Array{Auto,1}, ord::Int=ordenTaylor)
    N = length(coches)
    vec0T = [ Taylor([coches[i].x], ord) for i=1:N ]

    for k = 0:ord-1
        knext = k+1
        # Expansión de Taylor a orden k
        vv=Taylor[]
        for i=1:N
            xTt = Taylor( vec0T[i].coeffs[1:k+1], k)
            push!(vv,xTt)
        end

        # Las ecuaciones de movimiento

                XDot=Taylor[]
                for i=1:N
                    if i==N    #Para el último coche
                        numer=dist(vec0T[1].coeffs[1],vec0T[i].coeffs[1])-coches[1].c
                        if numer<coches[i].w      #Para el mínimo
                            XDott=dist(vv[1],vv[i])-coches[1].c
                        else
                            XDott=Taylor(coches[i].w,k)
                        end
                        push!(XDot,XDott)
                    else                #para los demás coches
                        numer = dist(vec0T[i+1].coeffs[1],vec0T[i].coeffs[1])-coches[i+1].c

                        if numer < coches[i].w
                            XDott=dist(vv[i+1],vv[i])-coches[i+1].c
                        else
                            XDott=Taylor(coches[i].w,k)
                        end
                        push!(XDot, XDott)
                    end
                end

        # De las ecuaciones de movimiento se tienen las recurrencias
        for i=1:N
            vec0T[i].coeffs[knext+1]=XDot[i].coeffs[knext] / knext
        end
    end

    return vec0T

end
```

Out[13]: jetTraffic (generic function with 2 methods)

¡¡¡Nuestro error!!!

In [14]:
```
#Aquí está el error

function TrafficIntegration( time_max::Float64, jetTraffic::Function, coches::Array{Auto,1}, orden::Int64=ordenTaylor,
                             epsilon::Float64=Epsilon  )
    #Condiciones iniciales

    t0 = 0.0
    coches = condiciones(N)
    N=lenght(coches)
    tV= Float64[]
    push!(tV, t0)

    R=Float64[]

    for i=1:N
        rr=mod2pi(coches[i].x+2pi)
        push!(R,rr)
    end

    H=Float64[]
    while t0<time_max && dt>0.2
        dt, coches = taylorStepper(jetTraffic, coches, ordenTaylor, Epsilon)
        t0 += dt
        @show t0
        coches

        push!(tV,t0)
        for i=1:N
            ss=coches[i].x
            push!(H,ss)
        end

        #v = Float64[]
        #w = Float64[]
  #      for i=1:5
   #         aux = coches[i]
    #        push!(v,cos(aux.x))
    #        push!(w,sin(aux.x))
     #   end
    end

    return R,H,coches
end
```

Out[14]: TrafficIntegration (generic function with 3 methods)

La solución

```
In [16]: a=rand(5)

Out[16]: 5-element Array{Float64,1}:
          0.937868
          0.993235
          0.60512
          0.545518
          0.426617

In [17]: M=Float64[]
         for i=1:5
             xi=1.1+i
            # @show xi
             push!(M,xi)
         end
         @show M

         M => [2.1,3.1,4.1,5.1,6.1]

Out[17]: 5-element Array{Float64,1}:
          2.1
          3.1
          4.1
          5.1
          6.1

In [18]: ao=hcat(a,[M])

Out[18]: 5x2 Array{Float64,2}:
          0.937868  2.1
          0.993235  3.1
          0.60512   4.1
          0.545518  5.1
          0.426617  6.1

In [19]: ao[1,2]

Out[19]: 2.1
```

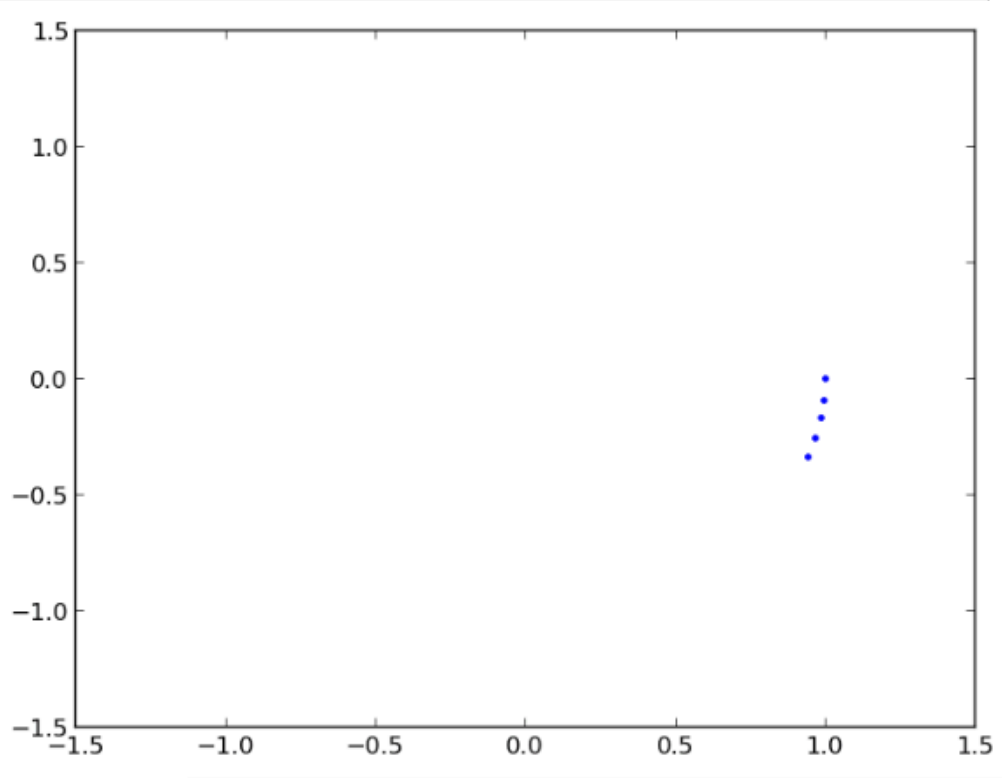
De forma retrógrada ☹

Como el integrador aún tiene detalles que arreglar, haremos paso a paso algunos valores de la integración

In [20]:
```
t0 = 0.0
coches = condiciones(5);
```

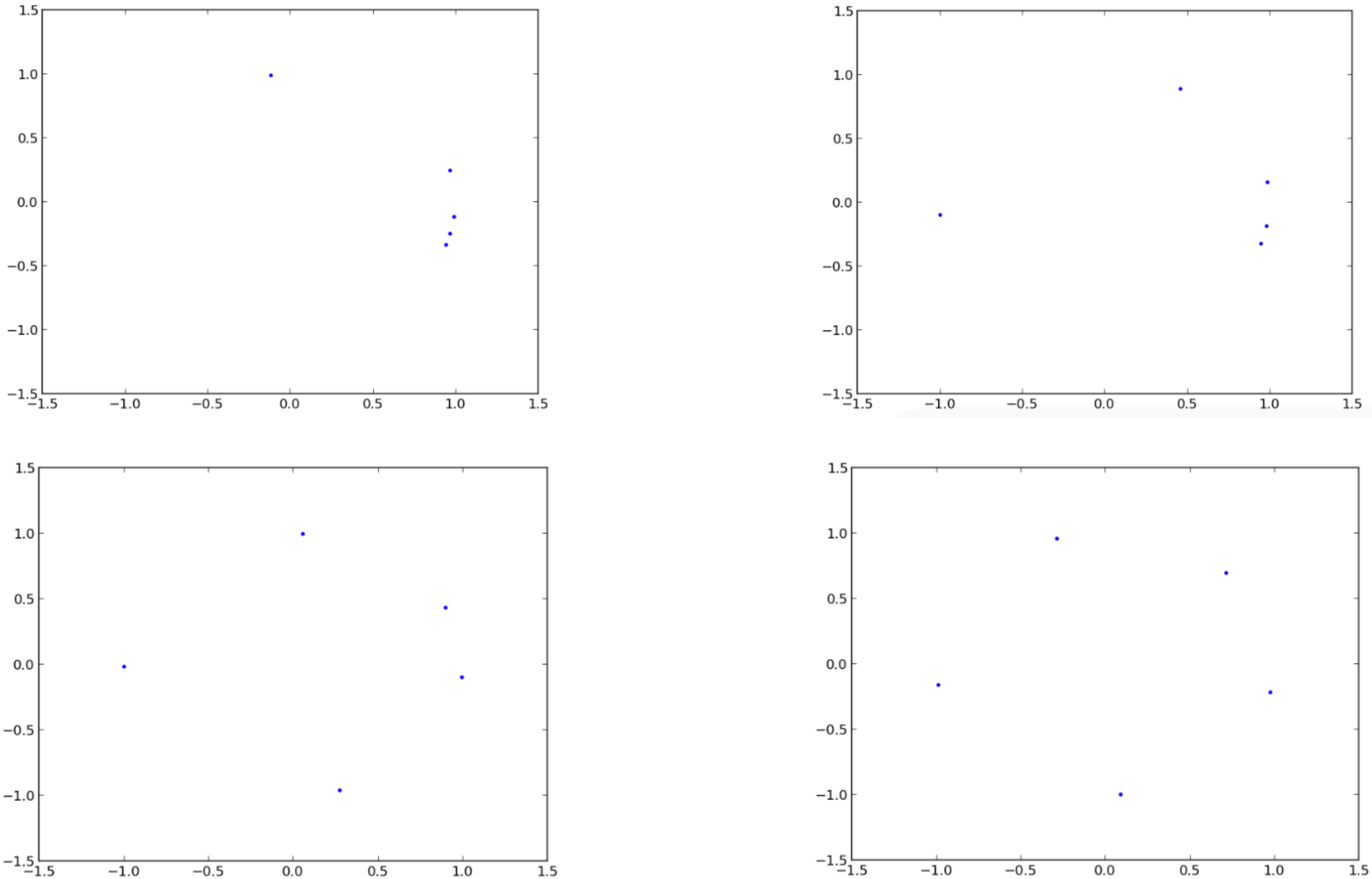
In [21]:
```
#A t=0
v0 = Float64[]
w0 = Float64[]
for i=1:5
    aux = coches[i]
    push!(v0,cos(aux.x))
    push!(w0,sin(aux.x))
end
```

In [22]:
```
plot(v0,w0,"b.")
axis([-1.5,1.5,-1.5,1.5])
```

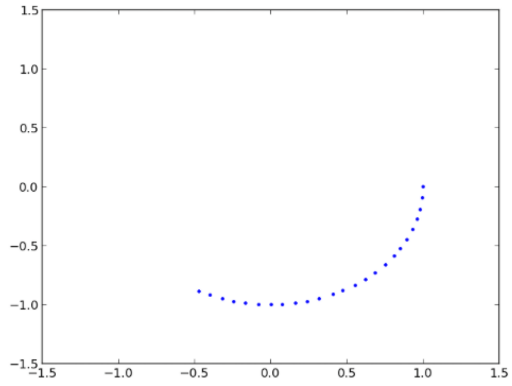


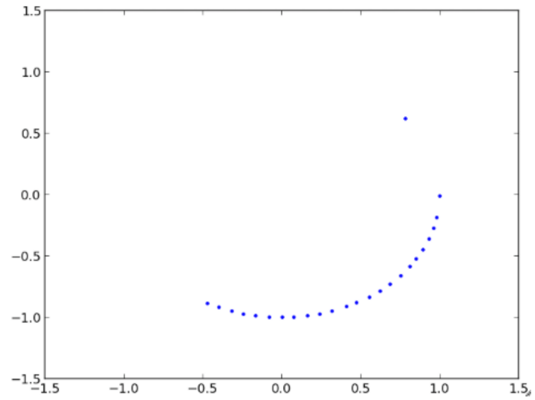
Para t's > 0

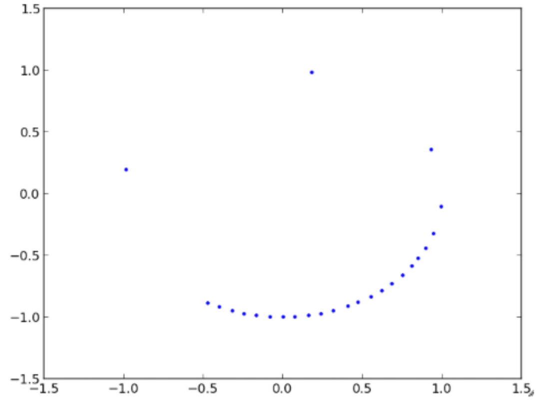
In [43]:
```
dt, coches = taylorStepper(jetTraffic, coches, ordenTaylor, Epsilon)
t0 += dt
#@show t0

coches;
```

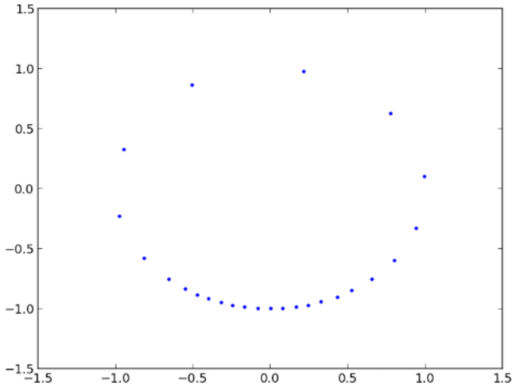


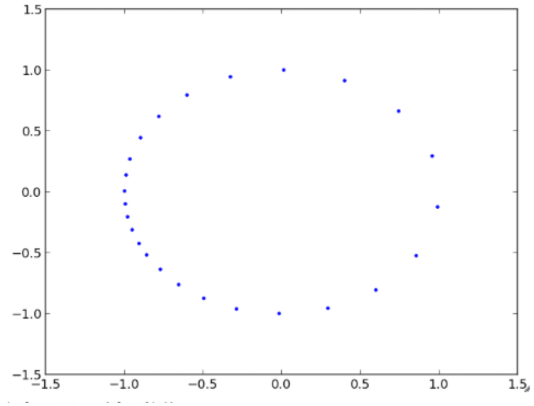
Para más coches (25)

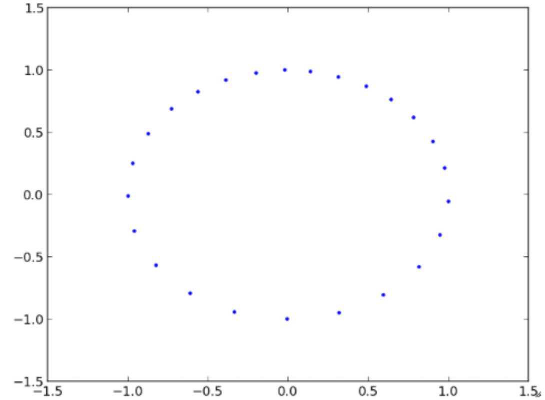




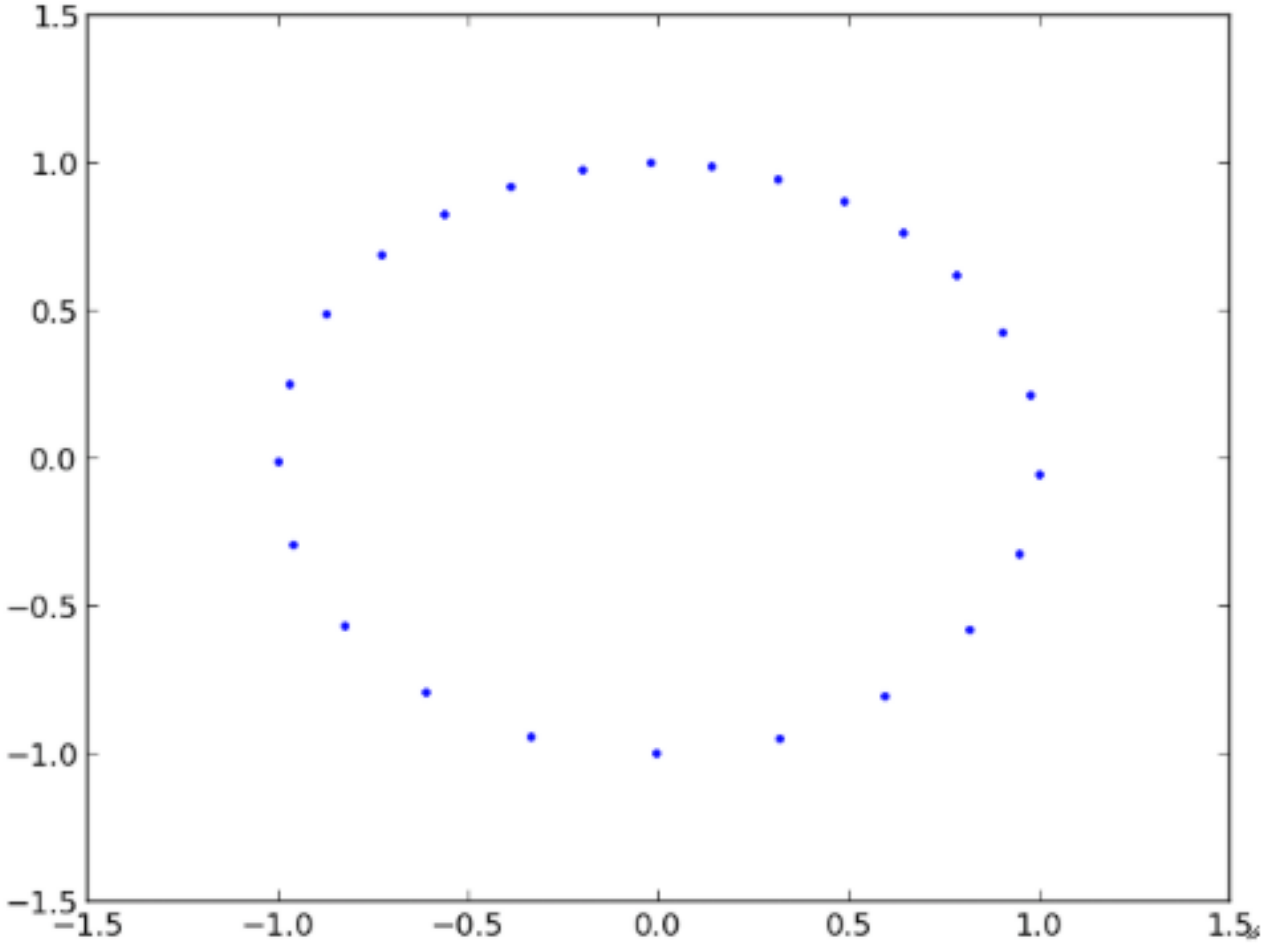








El equilibrio



Gracias !!!