7600017
Introdução à Física Computacional

Projeto 5: Dinâmica Populacional

Prof: José A. Hoyos

Victor Foscarini Almeida
nUsp: 10728101

São Carlos, 2019

Introdução

Nesse projeto será discutido acerca de sistemas dinâmicos e caos. Um sistema dinâmico é um sistema que evolui no tempo a partir de uma regra bem definida, que pode ser expressa na forma de uma equação, sendo que tal regra não muda com o tempo, itera-se essa mesma equação várias vezes e observa-se o desenvolvimento do sistema. No caso de uma função iterada, toma-se um valor inicial que é então iterado a partir de uma equação e, então, toma-se o resultado dessa primeira iteração e itera-se esse resultado, obtendo-se o resultado da segunda iteração. Esse processo é repetido tantas vezes quanto se queira, obtendo a partir dela a chamada órbita do sistema, que mostra a evolução dele com o tempo.

Um exemplo de sistema dinâmico é o crescimento de uma espécie isolada na ausência de predadores:

$$N_{i+1} = rN_i(1 - \frac{Ni}{Nmax})$$

Onde, pode-se chamar $\frac{Ni}{Nmax}$ de x_i e fazer o análogo com x_{i+1}, obtendo assim outra forma da equação:

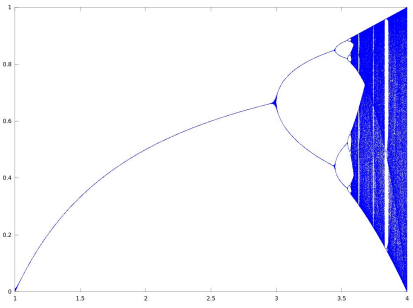
$$x_{i+1} = rx_i(1 - x_i)$$

Note que como estamos falando do crescimento de uma população, x sempre deve ser maior do que zero e menor do que 1, visto que apenas nesse caso será estudado o crescimento de uma população e será obtida uma solução para essa equação.

Um conceito fundamental acerca desse sistema é o de ponto fixo e, junto com ele, o de estabilidade. O ponto fixo é o ponto em que o sistema está estacionário e a sua evolução independe do tempo. Dependendo do valor de “r”, o sistema pode ter 1,2,4,8 e assim por diante, pontos fixos. Note que o período de ponto fixo(número de pontos fixos) é sempre dobrado.

Uma forma interessante de observar o sistema dinâmico acima é a partir do mapa logístico. Nele, no eixo X é representado o valor do parâmetro proporcional à taxa de crescimento populacional(r) e o eixo Y representa o valor ou os valores oscilantes que xi tende a obter após muitas iterações(período de ponto fixo), independentemente do valor inicial.

Assim, o mapa logístico e o diagrama de bifurcação são formas muito boas de se estudar o desenvolvimento de sistemas dinâmicos e representar o desenvolvimento de uma população, sendo muito utilizado para simular sistemas biológicos e, em alguns casos, como o sistema tende ao caos, até mesmo para obter-se números pseudo aleatórios.



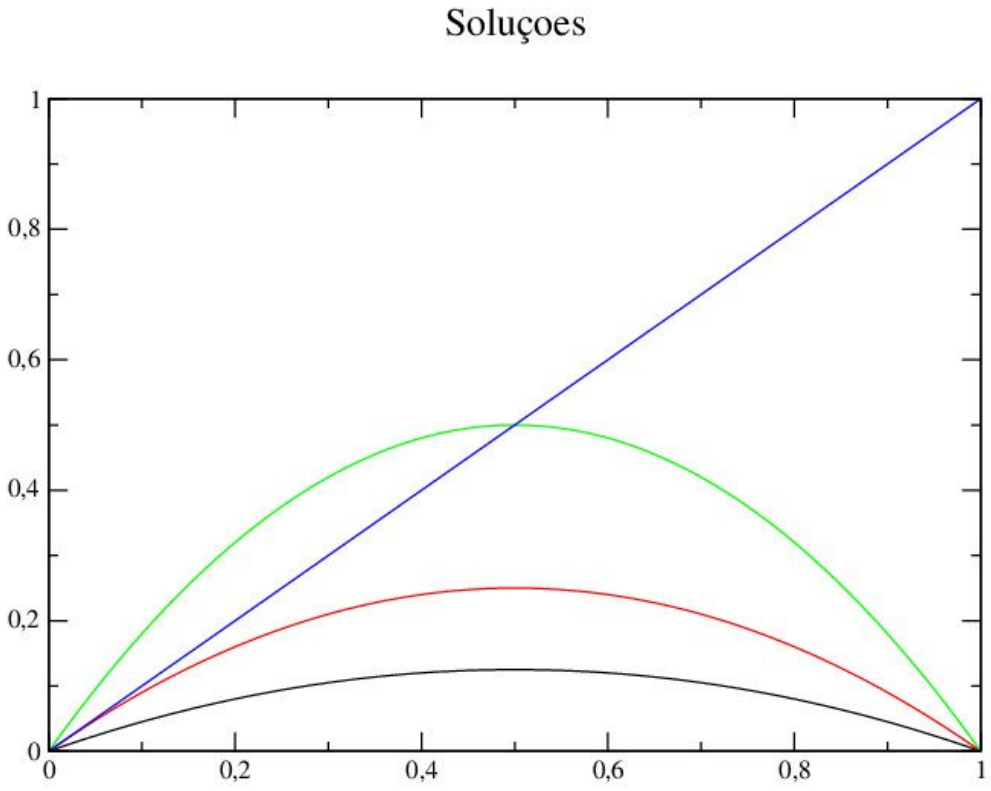
Desenvolvimento e Resultados

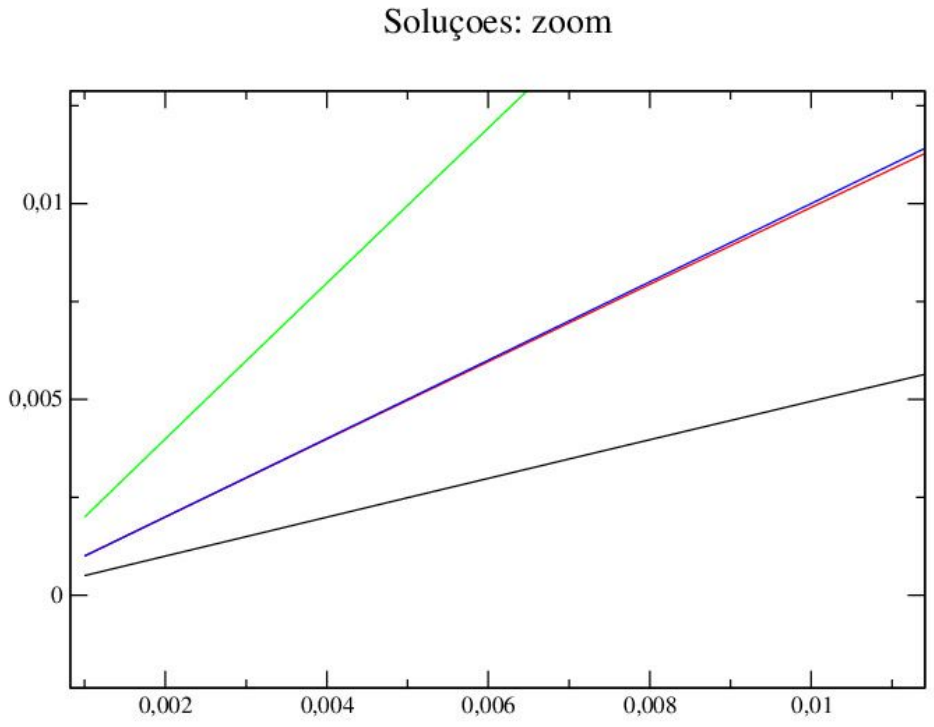
1 – Ponto fixo de período um

1.1 – Solução não trivial

É possível encontrar de forma analítica os valores possíveis de r para que se tenha uma solução para x = r*x*(1-x) de forma analítica. Resolvendo essa equação para x diferente de zero obtemos: $x = 1 - \frac{1}{r}$, assim como mostrado no projeto. Sabendo então, pela teoria do que o modelo representa, que x deve estar entre 0 e 1 e que r deve ser positivo para que x mantenha-se positivo durante as iterações, notamos que o valor de 1/r deve ser entre 0 e 1, logo, o valor de r deve ser maior do que 1.

Plota-se então os gráficos f(x) = x e g(x,r) = rx(1-x) para os valores de **r = 0.5**(em preto), **r=1**(em vermelho) e **r=2**(em verde).





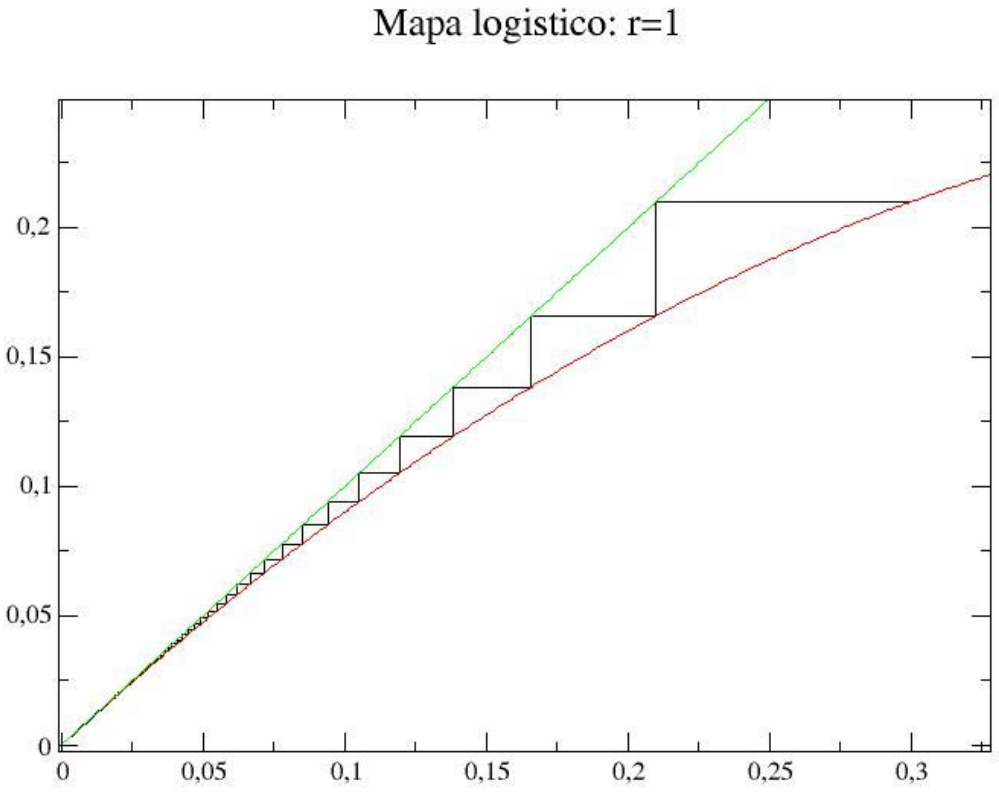

```
program solucoes

implicit none

integer i
real*8 r1,r2,r3,G

open(10,file=',5.dat')
open(20,file='1.dat')
open(30,file='2.dat')
open(40,file='x.dat')

r1 = 0.5d0
r2 = 1.d0
r3 = 2.d0

do i=1,999
      G = r1*real(i)*0.001d0*(1-real(i)*0.001d0)
      write(10,*)i*0.001d0,G
      G = r2*real(i)*0.001d0*(1-real(i)*0.001d0)
      write(20,*)i*0.001d0,G
      G = r3*real(i)*0.001d0*(1-real(i)*0.001d0)
      write(30,*)i*0.001d0,G
```

```
      write(40,*)i*0.001d0,i*0.001d0
enddo

end program
```

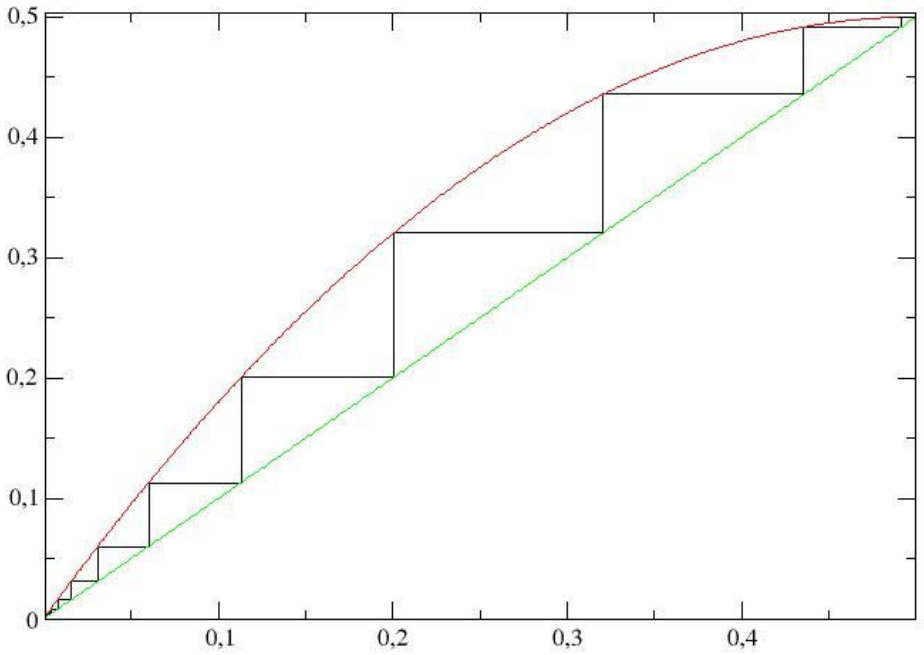
1.2 - Mapas Logísticos

Uma forma interessante de se observar a evolução de um sistema dinâmica com “r” fixo é a partir do mapa logístico. Plota-se, num gráfico, o valor de xi+1 em função de xi, sendo possível assim observar a trajetória e convergência para um valor de r, sendo que nos casos em que é possível (r>1) o valor converge para o ponto fixo.

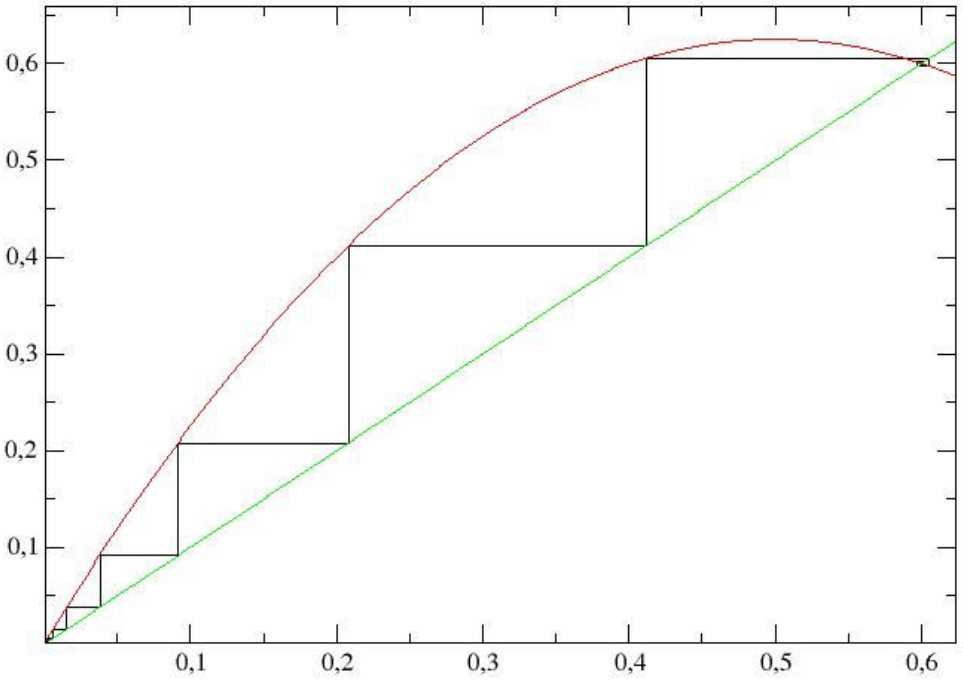
Note que a trajetória é limitada acima pela reta y=x e abaixo pela parábola encontrada no item anterior.



Mapa logistico: r=2

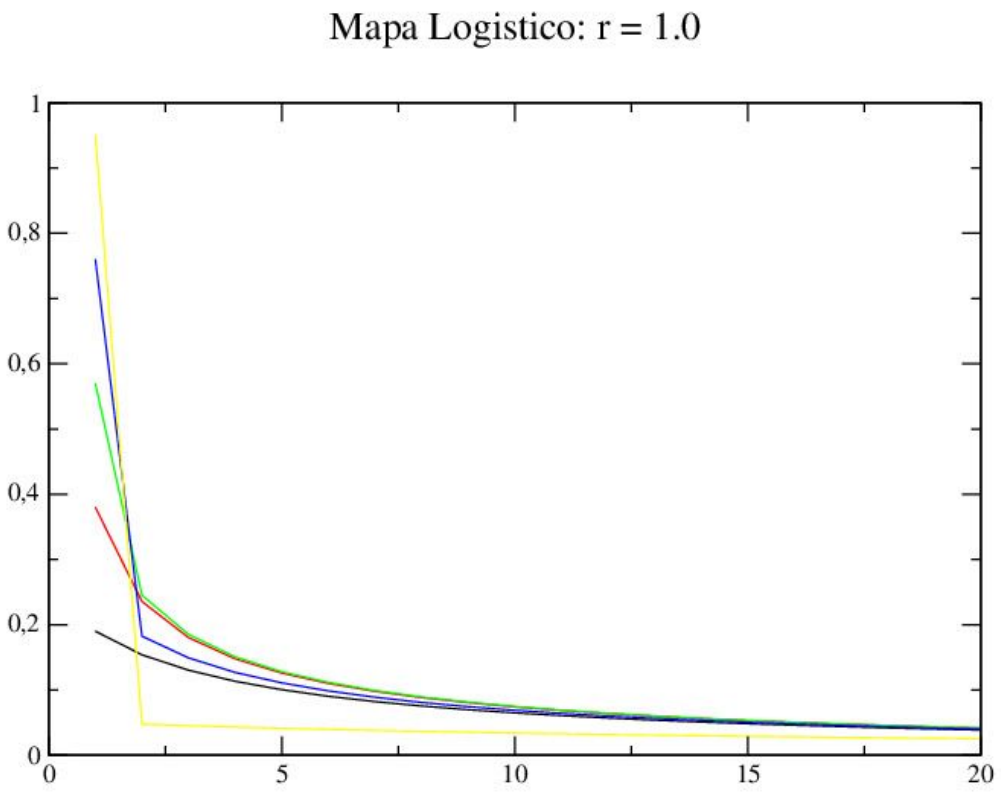


Mapa logistico: r=2.5

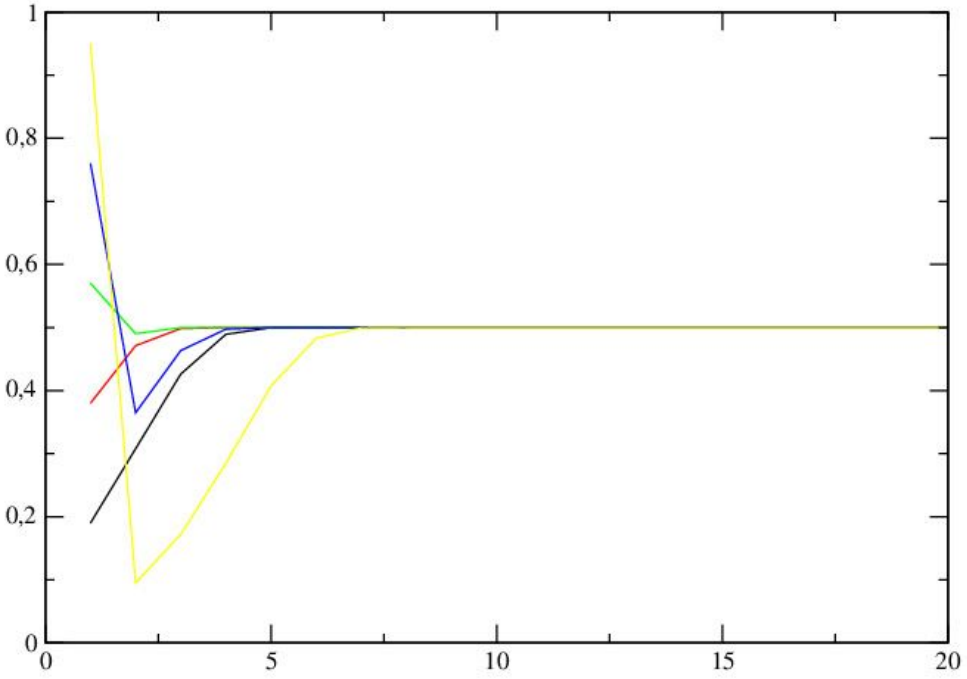


Abaixo estão os gráficos de “x” em função do número de iterações para os três valores de “r” pedidos no projeto.

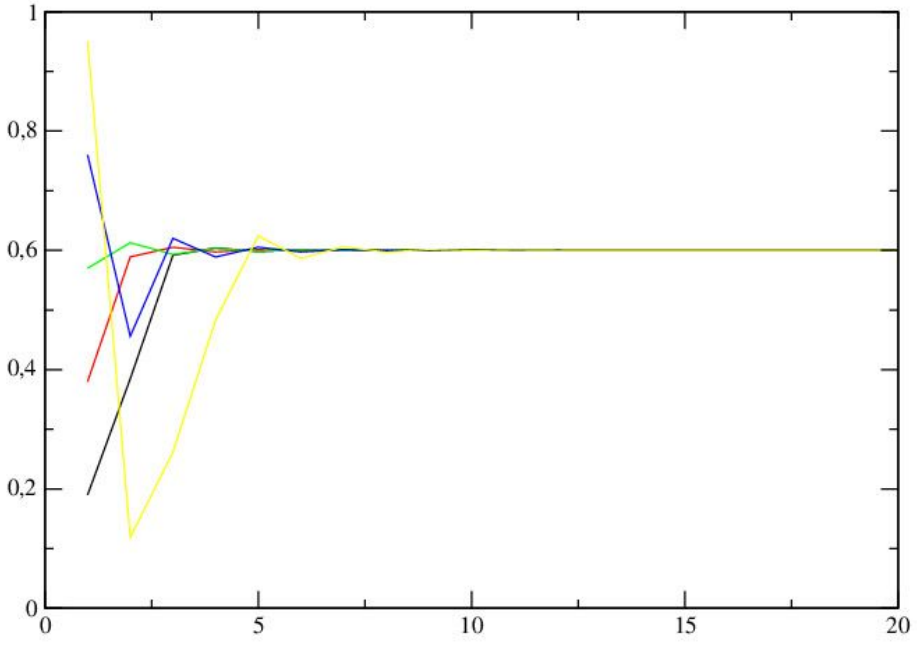
Note que eles foram erroneamente chamados de mapa logístico, porém são apenas um gráfico comum. Observou-se que para diferentes valores de X0, levou-se mais ou menos tempo para chegar ao ponto fixo, mas todos os valores convergiram no final.



Mapa Logistico: r = 2.0



Mapa Logistico: r = 2.5



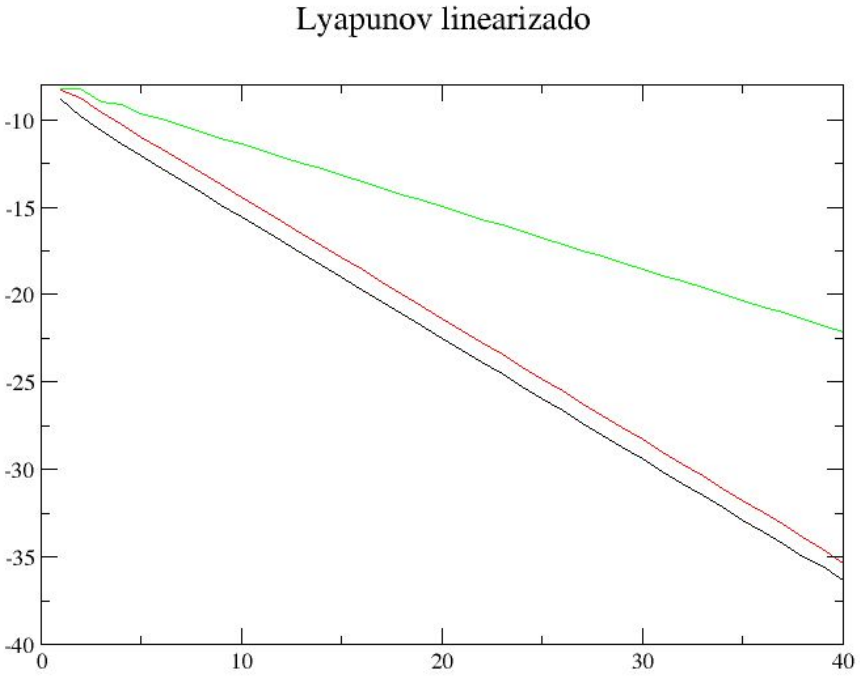
```
program mapalogistico

implicit none

integer*8 :: i,j
real*8 :: x,G
real*8, dimension(4) :: r
character :: c(1)

r(1) = 1.d0
r(2) = 2.d0
r(3) = 2.5d0
r(4) = 3.7d0

do i = 1,4

    c(1) = char(48+i)

    open(10, file = "mapalogistico"//c(1)//".dat")
    x = 0.3d0
    do j = 1,300
     G = r(i)*x*(1-x)
        write(10,*)x,G
     x = G
     write(10,*)x,x
    enddo

enddo

end program
```

```
program graficos

implicit none

integer i,j
real*8 r1,r2,r3,x1,x2,x3

r1 = 1.d0
r2 = 2.d0
r3 = 2.5d0
```

```
x1 = 0.d0
x2 = 0.d0
x3 = 0.d0

open(10,file='mapalogistico1.dat')
open(20,file='mapalogistico2.dat')
open(30,file='mapalogistico3.dat')

do j=1,5

      x1 = 0.19d0*j
      x2 = 0.19d0*j
      x3 = 0.19d0*j

      do i=1,20
            write(10,*)i,x1
            x1 = r1*x1*(1-x1)
            write(20,*)i,x2
            x2 = r2*x2*(1-x2)
            write(30,*)i,x3
            x3 = r3*x3*(1-x3)


      enddo

      write(10,*)
      write(20,*)
      write(30,*)

enddo

end program
```

1.3 - Distância ponto-a-ponto e expoente de Lyapunov

Aplicando a equação que gera o sistema dinâmico para os três valores de “r” pedidos, dado um valor inicial de “x” de 0.5 e um “x+epsilon” de 0.501 e calculando as distâncias, obtém-se as seguintes retas linearizadas:



a1,5 = -0.69558
a2,5 = -0.69366
a2,7 = -0.35766

A partir dos coeficientes lineares, obtemos o expoente de lyapunov, onde nota-se que para r=1,5 e r=2,5 o valor do coeficiente é semelhante, de cerca de -0,69 , já para r=2,7, o valor do coeficiente é de -0,35766.

```
program distancialyapunov

implicit none

integer i
real*8 r1,r2,r3,a1,a2,b1,b2,c1,c2,d1,d2,d3
```

```
!note que nesse codigo as variaveis a,b e c se referem ao chamado
"x" da equação que gera o sistema dinâmico
!d se refere às distâncias

r1 = 1.5d0
r2 = 2.5d0
r3 = 2.7d0


open(10,file='L1,5.dat')
open(20,file='L2,5.dat')
open(30,file='L2,7.dat')


a1 = 0.50d0
a2 = 0.51d0
b1 = 0.50d0
b2 = 0.51d0
c1 = 0.50d0
c2 = 0.51d0

do i=1,40

      a1 = r1*a1*(1.d0-a1)
      a2 = r1*a2*(1.d0-a2)
      d1 = abs(a2-a1)
      write(10,*)i,log(d1)

      b1 = r2*b1*(1.d0-b1)
      b2 = r2*b2*(1.d0-b2)
      d2 = abs(b2-b1)
      write(20,*)i,log(d2)

      c1 = r3*c1*(1.d0-c1)
      c2 = r3*c2*(1.d0-c2)
      d3 = abs(c2-c1)
      write(30,*)i,log(d3)

enddo

end program
```

1.4 - Formalização do expoente de Lyapunov

Note que a derivada de $G(x) = r \cdot x(1-x)$ é $G'(x) = r \cdot (1 - 2x)$, o que foi utilizado no código.

Obtém-se então, os seguintes resultados de expoente de Lyapunov para os seguintes valores de x inicial x0:

x0:	0.19000000000000000	lyapunov: -0.69233551029409723
x0:	0.38000000000000000	lyapunov: -0.70091327232845035
x0:	0.57000000000000006	lyapunov: -0.69944964024539658
x0:	0.76000000000000001	lyapunov: -0.70736602408161142
x0:	0.94999999999999996	lyapunov: -0.69473727806686214

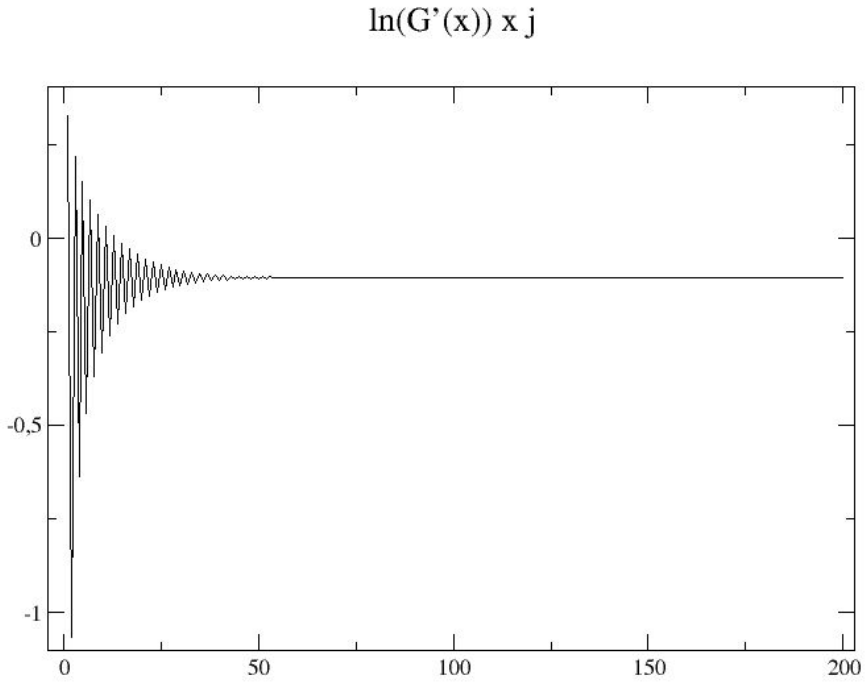
Os resultados corroboram com o obtido no item 1c, obtendo-se a partir da média que o expoente de Lyapunov é **-0.698 +/- 0.005**. Note que o valor obtido por esse método considerando a derivada é semelhante ao obtido a partir dos gráficos linearizados.

```
program distancialyapunov

integer i
real*8 r,x0,x,lambda

!note o "x" da equação que gera o sistema dinâmico
!d se refere às distâncias

r = 2.5d0

open(10,file='lambdas.dat')


do j=1,5
    x = 0.19d0*real(j)

    x0 = x !armazena o valor inicial de x

    do i=1,100

      x = r*x*(1.d0-x)
      lambda = lambda + log(abs(r*(1-2.d0*x)))

    enddo
```

```
    lambda = lambda/100.d0
    write(10,*) "x0:",x0,"lyapunov:",lambda

enddo

end program
```

1.5 - Expoente de Lyapunov para r=2.9

Fazendo um gráfico do expoente de lyapunov obtido a partir do ln(G'(x)) obtém-se:



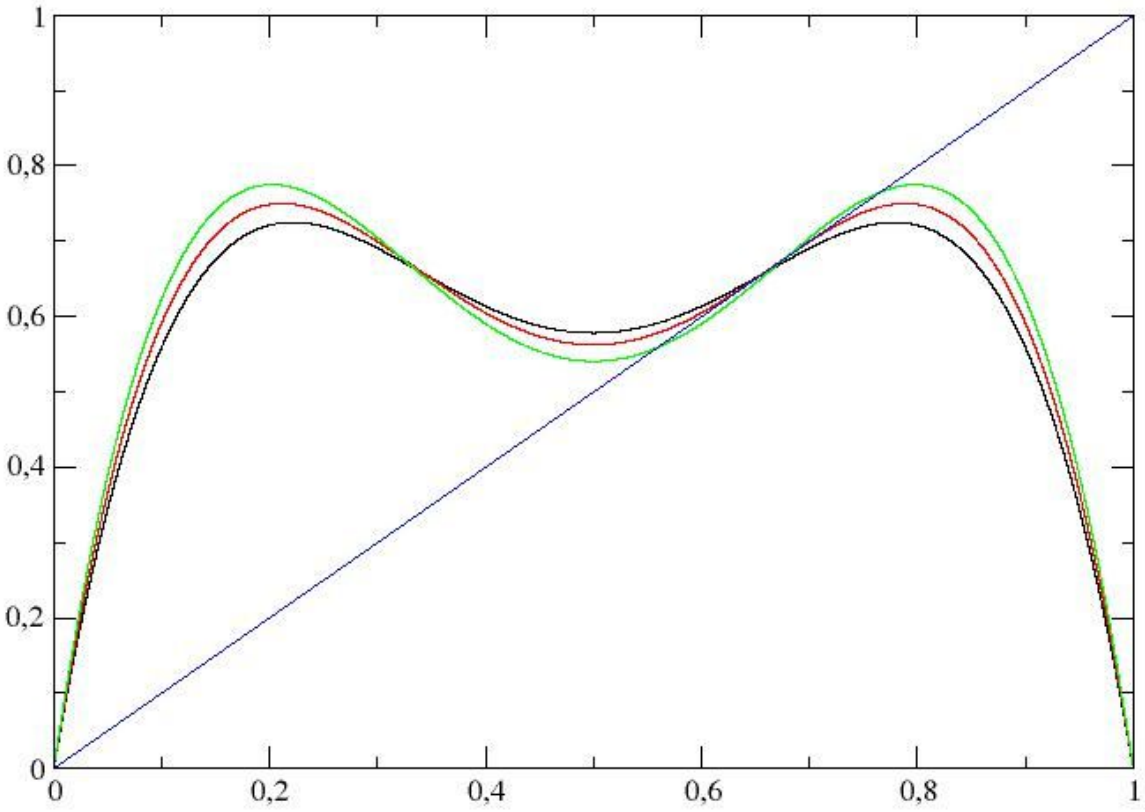
Nota-se que ao se aproximar de 200, quanto maior for o valor de "j"(número de iterações), o ln(G'(x)) converge para um valor específico. Tomando os valores para j>100 e calculando a média e o desvio padrão, o resultado obtido é de **-0.105360 +/- 0.000001**.

```
program distancialyapunov

integer i
real*8 r,x,lambda

!note o "x" da equação que gera o sistema dinâmico
!d se refere às distâncias

r = 2.9d0

open(10,file='lambdaXi.dat')

x = 0.1d0

do i=1,200

      x = r*x*(1.d0-x)
      lambda = log(abs(r*(1-2.d0*x)))

      write(10,*)i,lambda

enddo


end program
```

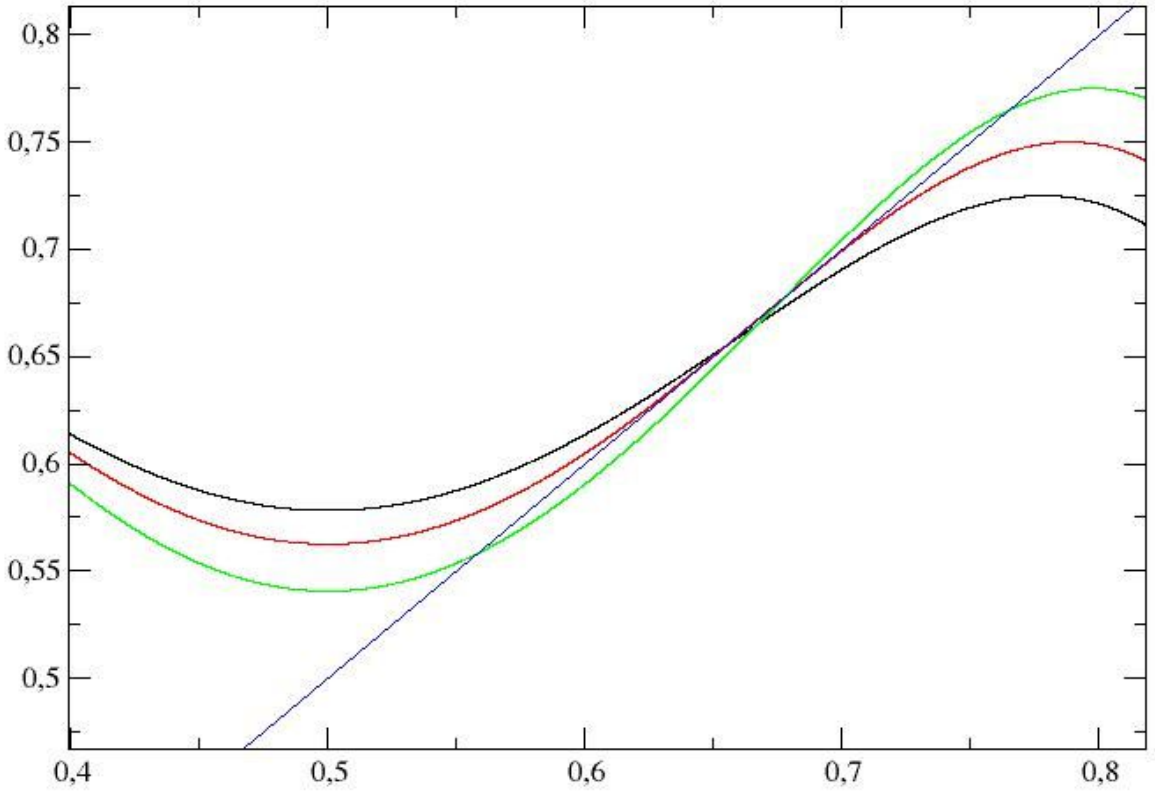
2- Dobras de período e caos

2.1 - Traçando o gráfico

Traçando o gráfico, pode-se notar que tem-se quatro intersecções para “r” valendo **3.1,** diferentemente dos valores anteriores de **2.9** e **3.0** . Note que 3.1 está em verde, 3.0 em vermelho e 2.9 em preto.



Dando zoom no gráfico:

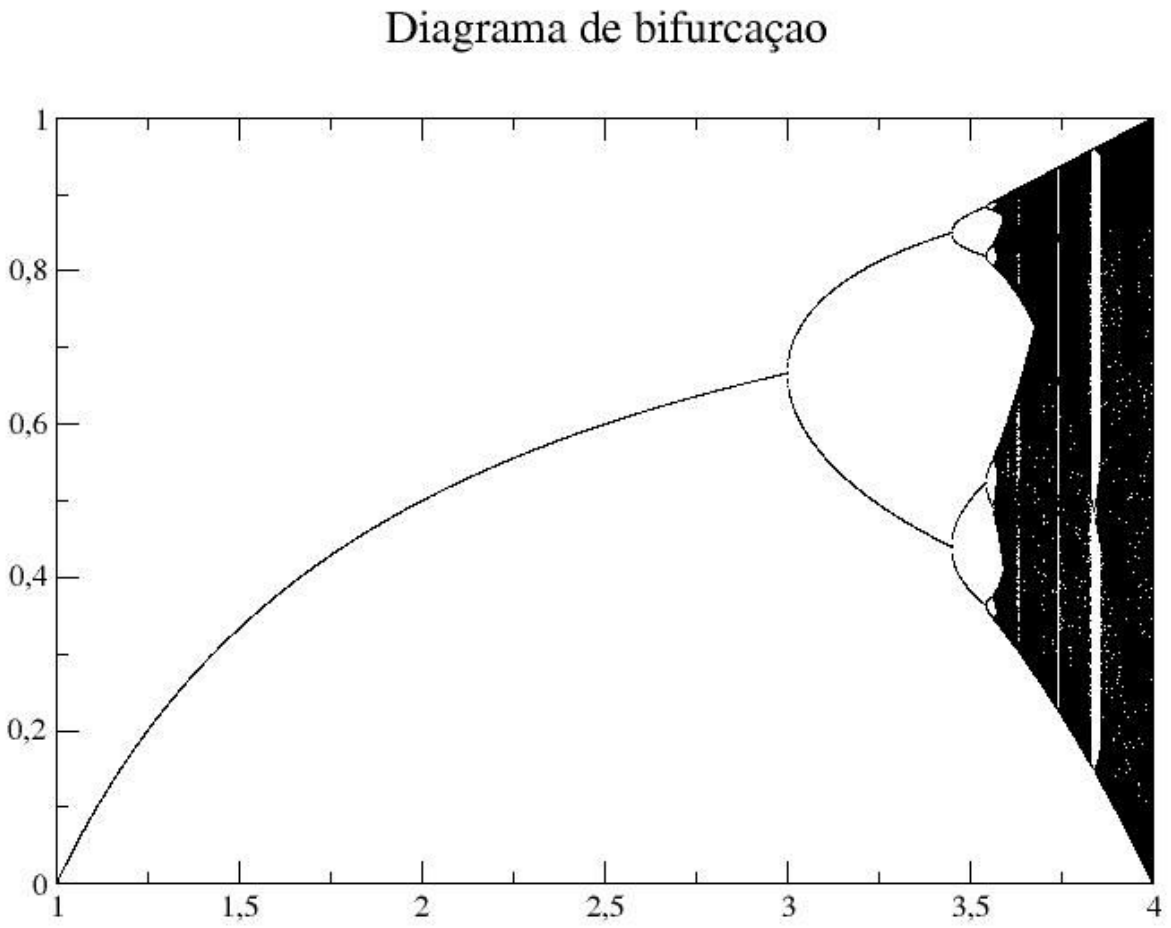



```
program grafico2

implicit none

integer :: i,j
real*8 :: x,r(3)
character :: c(1)
real*8,external :: G

r(1) = 2.9d0
r(2) = 3.d0
r(3) = 3.1d0

do j=1,3

    c(1) = char(48+j)
    open(10,file="graf2("//c(1)//").dat")
```

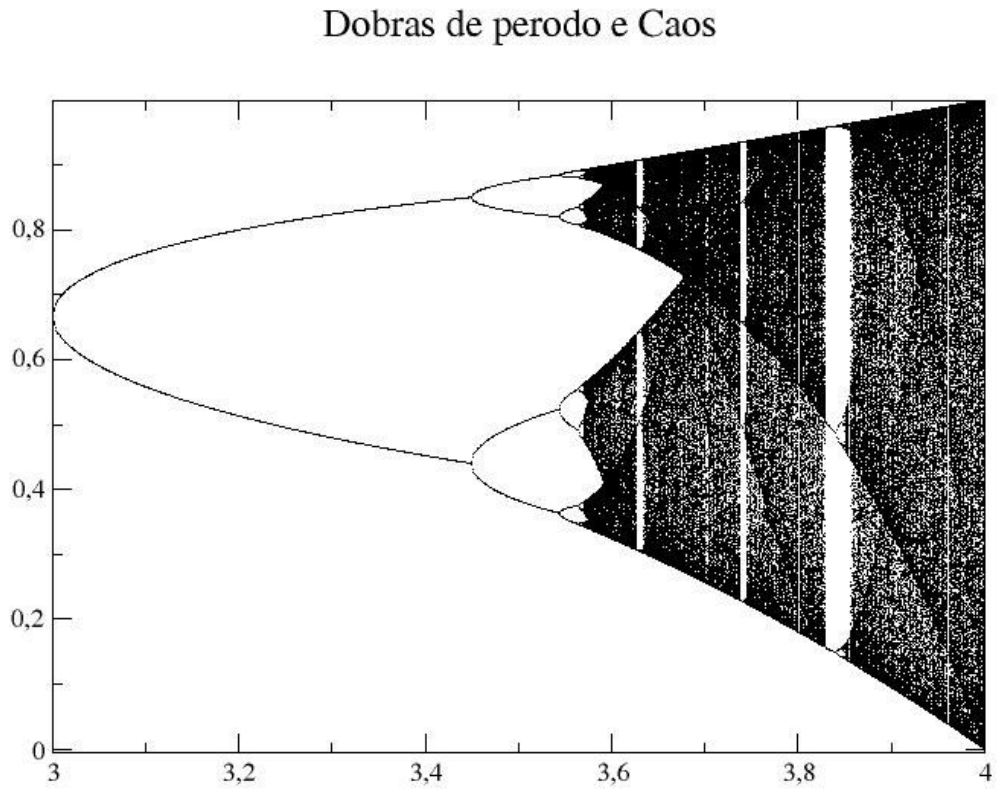
```
    do i=0,10**6
        x = real(i,8)/float(10**6)
        write(10,*) x,G(G(x,r(j)),r(j))
    enddo

    close(10)

enddo

open(20,file="graf2y=x.dat")

do i=1,10**6

    x = i/(10**6)

    write(20,*)x,x

enddo

end program

function G(x,r)
    implicit none
    real*8 :: x,r
    real*8 G

    G = x*r*(1.d0-x)

    return
end function
```

2.2 - Diagrama de bifurcação

O diagrama de bifurcação é uma forma interessante de se analisar o desenvolvimento de sistemas que seguem a equação G(x) = r*x*(1-x).Nele, relaciona-se cada valor de “r” com os seu(s) ponto(s) fixo(s).



Dando zoom na parte mais interessante do diagrama onde o número de pontos fixos começa a dobrar (fenômeno conhecido como dobra de período) e o sistema tende ao caos:




```
program diagramabifurcacao

!varia-se o r de 1.0 até 4.0 de 0.1 em 0.1, e para cada e,
itera-se o x 1000 vezes para chegar no que seria(m) o(s) pontos
fixo(s) e, então, printa-se num arquivo os valores de r e do(s)
ponto(s) fixo(s) encontrado(s) também 1000 vezes, para garantir
que obtenha-se um valor próximo do real

implicit none

integer i,j
real*8 r,x

open(10,file='bifurcação.dat')

x = 0.5d0

do k=1,3000
```

```
      r = 0.001d0 * real(k) + 1.d0


      do i=1,1000
            x = r*x*(1-x)
      enddo

      do j=1,500
          x = r*x*(1-x)
          write(10,*) r,x
      enddo

enddo

end program
```

2.3 - Constante de Feigenbaum

Utilizando o mapa de bifurcação do item anterior, encontra-se os valores de r1(onde ocorre a primeira dobra de período tendo dois pontos fixos), r2(segunda dobra, quatro pontos fixos) e r3(terceira dobra, três pontos fixos).

r1 = 3.00
r2 = 3.45
r3 = 3.54

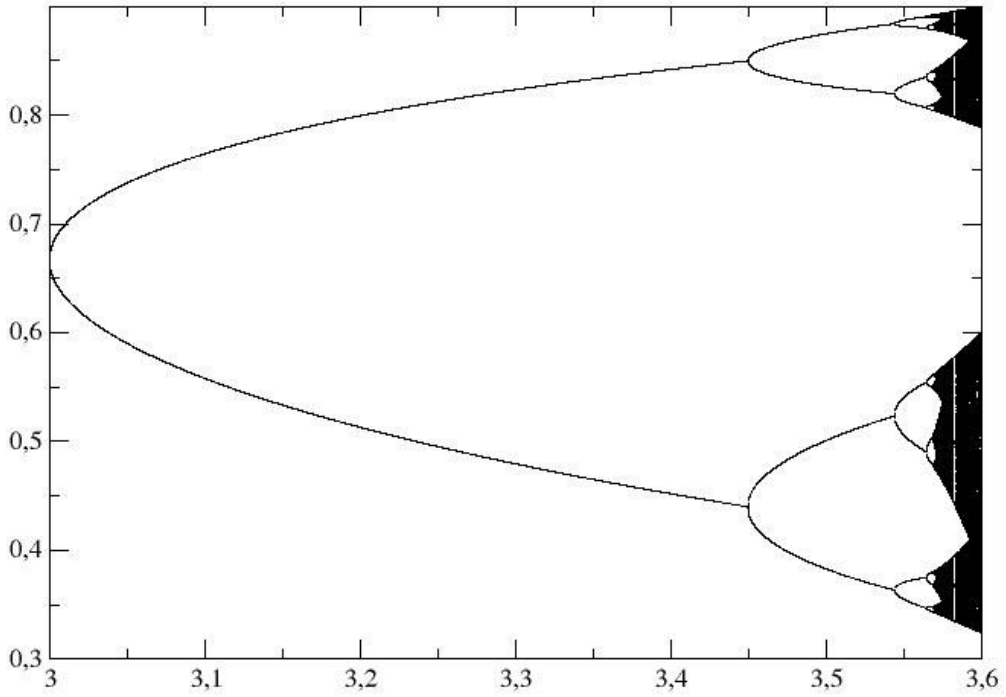
A partir da equação dada no projeto, encontra-se então o valor de **5,00** para a constante de Feigenbaum**.** Nota-se um valor próximo do esperado de 4,669 (fonte: Briggs, Keith (July 1991). "A Precise Calculation of the Feigenbaum Constants"), porém com um erro considerável de precisão, o que mostra que a necessidade de mudar o código para ter uma precisão maior e reduzir o número de valores de r(para o código ser mais leve).

Alterando o código anterior então e tomando apenas r entre 3.000 e 3.600, com um menor valor de epsilon(variação de r no código), são obtidos os seguintes valores:

r1 = 3.000
r2 = 3.449
r3 = 3.554

Com esses valores, obtemos então a constante de Feigenbaum de **4.726,** um valor mais próximo do esperado. É razoável então supor que se fosse aumentada

ainda mais a precisão do código, seria possível chegar a um valor igual ao esperado.



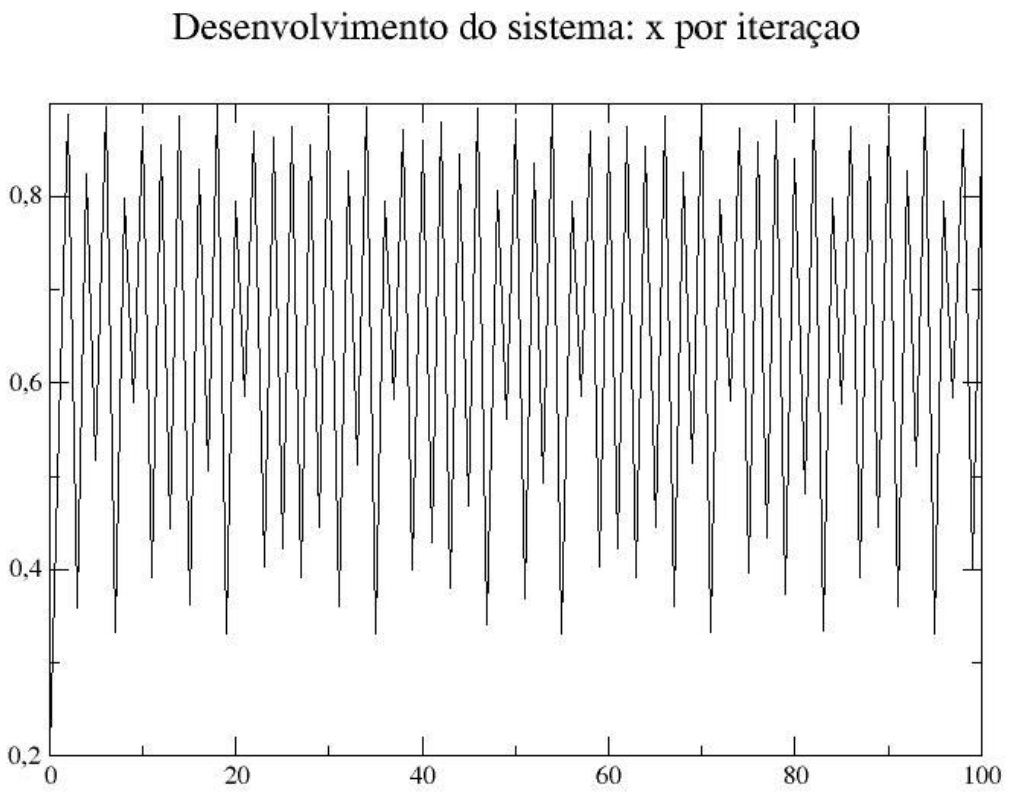

```
program mapabifurcacao

!varia-se o r de 1.0 até 4.0 de 0.1 em 0.1, e para cada e,
itera-se o x 1000 vezes para chegar no que seria(m) o(s) pontos
fixo(s) e, então, printa-se num arquivo os valores de r e do(s)
ponto(s) fixo(s) encontrado(s) também 1000 vezes, para garantir
que obtenha-se um valor próximo do real

integer i,j
real*8 r,x

open(10,file='bifurcação.dat')

x = 0.5d0

do k=1,6000 !modifica-se aqui para uma precisão maior e apenas o
intervalo interessante para o problema da 3c

      r = 0.0001d0 * real(k) + 3.d0
```

```
      do i=1,1000
            x = r*x*(1-x)
      enddo

      do j=1,500
          x = r*x*(1-x)
          write(10,*) r,x
      enddo

enddo

end program
```

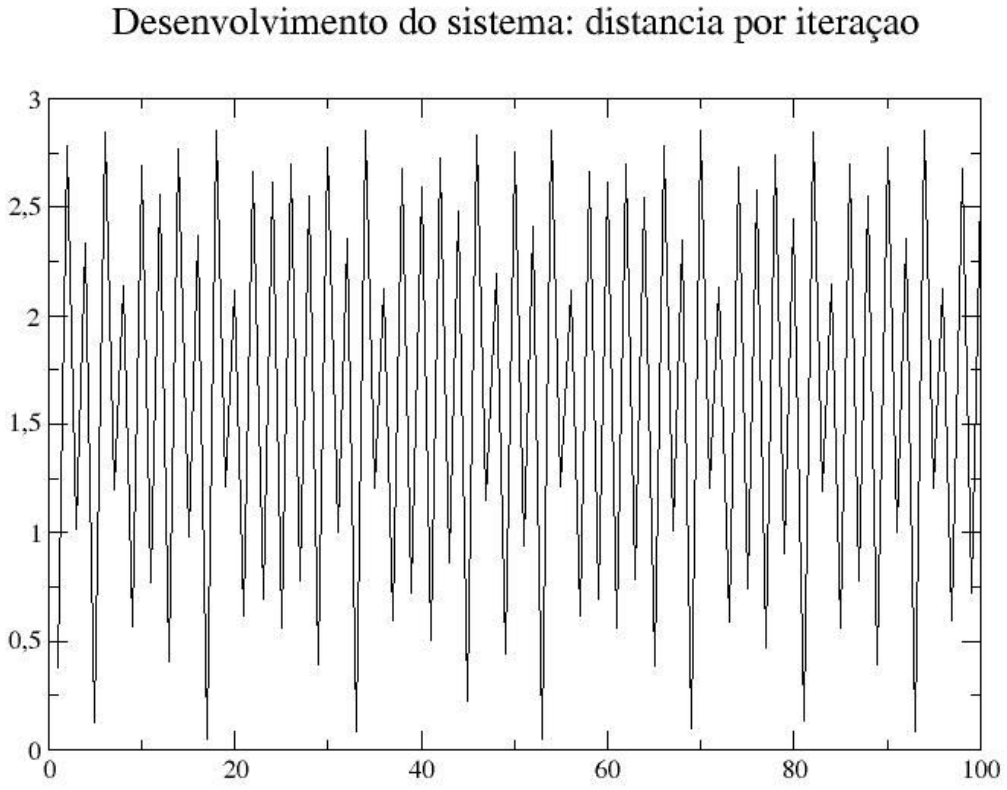
2.4 - Caos Determinístico

Dessa vez, em um único código foi possível, a partir de 5 valores de “r” diferentes entre 3.58 e 4, analisar o desenvolvimento do sistema em cada iteração a partir 5 diferentes valores de x0 inicial(note que no código foram salvos em arquivos com nome “caos”), além da distância entre os valores de x em cada iteração para um x0 igual a x somado a um epsilon(no código salvos em arquivos com nome dist). Por fim, foi possível analisar os coeficientes de lyapunov obtidos nesta conta, obtendo para cada valor de r cinco coeficientes de lyapunov(relativos a cada valor de x0 diferente).

Para o desenvolvimento do sistema, analisando o valor de x para cada iteração, foram obtidos gráficos no seguinte formato:



Já para a distância em função da iteração, obteve-se algo assim:



Como para qualquer valor no intervalo obtém-se gráficos semelhantes, optou-se por colocar apenas os gráficos para o valor de r=3.59 e de x0 = 0.19, visto que para diferentes valores de x0 obteve-se um gráfico semelhante.

Já para o cálculo do coeficiente de lyapunov, imprimiu-se num arquivo os valores encontrados e a cópia do texto contido neste arquivo é a seguinte:

r: 3.5899999999999999
x0,lyapunov
0.19000000000000000 0.14757963750593134
0.39000000000000001 0.14502989832280416
0.58999999999999997 0.15155326715804557
0.79000000000000004 0.15172297655744876
0.98999999999999999 0.19078385483866697

r: 3.6899999999999999
x0,lyapunov
0.19000000000000000 0.35805130402955726
0.39000000000000001 0.35730562371899183
0.58999999999999997 0.33924085919816144
0.79000000000000004 0.30956851385197898
0.98999999999999999 0.37526347401547894

r: 3.7900000000000000

x0,lyapunov
0.19000000000000000 0.39019621109939051
0.39000000000000001 0.41815063501893690
0.58999999999999997 0.38841679971594878
0.79000000000000004 0.42513534047584728
0.98999999999999999 0.43105903097962800

r: 3.8900000000000001
x0,lyapunov
0.19000000000000000 0.49426472701825908
0.39000000000000001 0.47141594550649091
0.58999999999999997 0.49594158642216668
0.79000000000000004 0.47723211543570898
0.98999999999999999 0.51631991950591571

r: 3.9900000000000002
x0,lyapunov
0.19000000000000000 0.65353886907241443
0.39000000000000001 0.66877334962259782
0.58999999999999997 0.68205680512997946
0.79000000000000004 0.62978374388459646
0.98999999999999999 0.65918013007818876

Note que não há uma diferença muito grande para cada valor do coeficiente de lyapunov para diferentes valores de x0, o que colabora com a hipótese de que os gráficos são semelhantes independente de x0.

O resultado mais interessante obtido neste item foi o caos obtido no desenvolvimento do sistema, tanto nos gráficos de x por iteração quanto nos gráficos da distância por iteração, mostrando que em um intervalo finito o sistema não converge para nenhum valor de x ou valores de pontos fixos.

```
program caos

implicit none

real*8,parameter :: eps=10**(-10)
real*8,dimension(5) :: x,x0,r,x0e,xe,d
real*8,dimension(5,5) :: lyapunov
integer*8 :: i,j,k


r(1) = 3.59d0
r(2) = 3.69d0
r(3) = 3.79d0
r(4) = 3.89d0
r(5) = 3.99d0

x0(1) = 0.19d0
```

```
x0(2) = 0.39d0
x0(3) = 0.59d0
x0(4) = 0.79d0
x0(5) = 0.99d0

do i=1,5
     x0e(i) = x0(i) + eps !note que x0e se refere a x somado com
epsilon
enddo


open(1,file="caos1.dat")
open(2,file="caos2.dat")
open(3,file="caos3.dat")
open(4,file="caos4.dat")
open(5,file="caos5.dat")

open(10,file="dist1.dat")
open(20,file="dist2.dat")
open(30,file="dist3.dat")
open(40,file="dist4.dat")
open(50,file="dist5.dat")

open(60,file="lyapunov.dat")

do i=1,5

     x(1) = x0(i)
     x(2) = x0(i)
     x(3) = x0(i)
     x(4) = x0(i)
     x(5) = x0(i)

     xe(1) = x0e(i)
     xe(2) = x0e(i)
     xe(3) = x0e(i)
     xe(4) = x0e(i)
     xe(5) = x0e(i)

     write(i,*)j,x(1),x(2),x(3),x(4),x(5)

     do j=1,100

          x(1) = r(1)*x(1)*(1.d0 - x(1))
          xe(1) = r(1)*xe(1)*(1.d0 - xe(1))
          d(1) = abs(r(1)*(1-2.d0*x(1)))
          lyapunov(1,i) = lyapunov(1,i) + log(d(1))

          x(2) = r(2)*x(2)*(1.d0 - x(2))
          xe(2) = r(2)*xe(2)*(1.d0 - xe(2))
          d(2) = abs(r(2)*(1-2.d0*x(2)))
```

```
            lyapunov(2,i) = lyapunov(2,i) + log(d(2))

            x(3) = r(3)*x(3)*(1.d0 - x(3))
            xe(3) = r(3)*xe(3)*(1.d0 - xe(3))
            d(3) = abs(r(3)*(1-2.d0*x(3)))
            lyapunov(3,i) = lyapunov(3,i) + log(d(3))

            x(4) = r(4)*x(4)*(1.d0 - x(4))
            xe(4) = r(4)*xe(4)*(1.d0 - xe(4))
            d(4) = abs(r(4)*(1-2.d0*x(4)))
            lyapunov(4,i) = lyapunov(4,i) + log(d(4))

            x(5) = r(5)*x(5)*(1.d0 - x(5))
            xe(5) = r(5)*xe(5)*(1.d0 - xe(5))
            d(5) = abs(r(5)*(1-2.d0*x(5)))
            lyapunov(5,i) = lyapunov(5,i) + log(d(5))

            write(i,*)j,x(1),x(2),x(3),x(4),x(5)
            write(10*i,*)j,d(1),d(2),d(3),d(4),d(5)


      end do

      lyapunov(1,i) = lyapunov(1,i)/100
      lyapunov(2,i) = lyapunov(2,i)/100
      lyapunov(3,i) = lyapunov(3,i)/100
      lyapunov(4,i) = lyapunov(4,i)/100
      lyapunov(5,i) = lyapunov(5,i)/100

end do

!salva-se num arquivo os valores obtidos dos coeficientes de
lyapunov

write(60,*)"r:",r(1)
write(60,*)"x0,lyapunov"
do k=1,5
      write(60,*)x0(k),lyapunov(1,k)
enddo
write(60,*)

write(60,*)"r:",r(2)
write(60,*)"x0,lyapunov"
do k=1,5
      write(60,*)x0(k),lyapunov(2,k)
enddo
write(60,*)

write(60,*)"r:",r(3)
write(60,*)"x0,lyapunov"
do k=1,5
```

```
      write(60,*)x0(k),lyapunov(3,k)
enddo
write(60,*)

write(60,*)"r:",r(4)
write(60,*)"x0,lyapunov"
do k=1,5
      write(60,*)x0(k),lyapunov(4,k)
enddo
write(60,*)

write(60,*)"r:",r(5)
write(60,*)"x0,lyapunov"
do k=1,5
      write(60,*)x0(k),lyapunov(5,k)
enddo
write(60,*)

end program
```

3 – Modelo predador-presa

É interessante montar modelos ecológicos para se estudar a relação entre o crescimento e decrescimento das populações de espécies em relações predatórias. É de conhecimento da ecologia que com o crescimento da população de presas, há uma tendência do aumento da população de predadores e com o decrescimento da população de presas há uma tendência à diminuição da população de predadores, estando a população de predadores ligada diretamente à existência de presas no ambiente. Já com respeito à população de predadores, ela é um limitante ao crescimento da população de presas. Assim, com o aumento da população de predadores há uma tendência à diminuição do número de presas, já com a diminuição da população de predadores, há uma tendência ao aumento da população de presas. Nota-se então, que as populações das duas espécies estão diretamente relacionadas, sendo uma em função da outra.

3.1 - Parâmetros a,b,c e d

Tem-se as seguintes equações para simular o modelo predador-presa, onde X se refere a presa e Y se refere ao predador:

$$\frac{dx}{dt} = ax - bxy$$
$$\frac{dy}{dt} = -cy + dxy$$

Note que os parâmetros **a** e **c** se referem, respectivamente, ao crescimento natural das presas e o decrescimento natural dos predadores. No caso das presas, que dependem apenas de si mesmas, não havendo predadores, elas se reproduzem e sua população tende a aumentar (note que esse modelo não considera os recursos naturais como um limitante), de acordo com o parâmetro **a**. Já no caso dos predadores, eles dependem diretamente das presas para sobreviver, então apresentam um decrescimento natural, que está ligado ao parâmetro **c**.

Já os parâmetros **b** e **d** se referem ao decrescimento das presas devido à interação com os predadores e ao crescimento dos predadores devido à sua interação com as presas. Sendo assim, representam a relação predatória entre as espécies onde o predador se beneficia em detrimento das presas, que são prejudicadas.

As unidades obtidas em **a** e **c** são **(ano)$^{-1}$** , já as unidades obtidas em **b** e **d** são **(ano*milhares de animais)$^{-1}$** .

3.2 - Método Runge-Kutta

No projeto anterior, já foi visto que o método Runge-Kutta é bem eficiente para se realizar integração numérica. Então, utilizou-se esse método para resolver as equações diferenciais.

Aplica-se, para cada objeto:

$$x_{i+1} = x_i + \frac{\Delta T}{6}(F_x^{(1)} + 2F_x^{(2)} + 2F_x^{(3)} + F_x^{(4)})$$
$$y_{i+1} = y_i + \frac{\Delta T}{6}(F_y^{(1)} + 2F_y^{(2)} + 2F_y^{(3)} + F_y^{(4)})$$

Onde, para x:

$$F_x^{(1)} = fx(x, y)$$
$$F_x^{(2)} = fx(x + 0.5 \cdot dT \cdot F_x^{(1)}, y + 0.5 \cdot dT \cdot F_y^{(1)})$$
$$F_x^{(3)} = fx(x + 0.5 \cdot dT \cdot F_x^{(2)}, y + 0.5 \cdot dT \cdot F_y^{(2)})$$
$$F_x^{(4)} = fx(x + dT \cdot F_x^{(3)}, y + dT \cdot F_y^{(3)})$$

E, para y:

$$F_y^{(1)} = fy(x, y)$$
$$F_y^{(2)} = fy(x + 0.5 \cdot dT \cdot F_x^{(1)}, y + 0.5 \cdot dT \cdot F_y^{(1)})$$
$$F_y^{(3)} = fy(x + 0.5 \cdot dT \cdot F_x^{(2)}, y + 0.5 \cdot dT \cdot F_y^{(2)})$$
$$F_y^{(4)} = fy(x + dT \cdot F_x^{(3)}, y + dT \cdot F_y^{(3)})$$

Onde:

$$fx(x, y) = a \cdot x - b \cdot x \cdot y$$
$$fy(x, y) = b \cdot x - c \cdot x \cdot y$$

```
program predadorpresaRK4
```

```
!código runge-kutta para solução do problema de predador-presa

!dx/dt = ax - bxy (presas)
!dy/dt = -cy +dxy (predadores)

    implicit none

    real*8,external :: fx,fy
    real*8 :: x,y,vx,vy
    real*8 :: kx(5),ky(5)
    real*8 :: T,dT,Tf
    integer :: N,i

open(10,file="presaXt.dat")
open(20,file="predadorXt.dat")
open(30,file="trajetoria.dat")

!as condições iniciais devem ser inseridas aqui

x =  !numero de presas
y =  !numero de predadores

!intervalo de iteração

dT = 0.001d0
Tf =  !numero de anos a serem iterados
N = int(Tf/dT)
T = 0.d0

!loop

do i=1,N

    !atualiza o tempo
    T = T + dT

    !atualiza as funções do tempo(X e Y)
    kx(1) = fx(x,y)
    ky(1) = fy(x,y)
    kx(2) = fx(x+0.5d0*dT*kx(1),y+0.5d0*dT*ky(1))
    ky(2) = fy(x+0.5d0*dT*kx(1),y+0.5d0*dT*ky(1))
    kx(3) = fx(x+0.5d0*dT*kx(2),y+0.5d0*dT*ky(2))
    ky(3) = fy(x+0.5d0*dT*kx(2),y+0.5d0*dT*ky(2))
    kx(4) = fx(x+dT*kx(3),y+dT*ky(3))
    ky(4) = fy(x+dT*kx(3),y+dT*ky(3))
```

```
    x = x + dT/6.d0*(kx(1) + 2*(kx(2)+kx(3)) + kx(4))
    y = y + dT/6.d0*(ky(1) + 2*(ky(2)+ky(3)) + ky(4))

    write(10,*)T,x
    write(20,*)T,y

enddo

end program


function fx(x,y)
   implicit none
   real*8 :: t,x,y
   real*8, parameter :: a=,b=,c=,d= !os parâmetros devem ser
inseridos aqui
   real*8 fx
   fx = a*x - b*x*y
   return
end function

function fy(x,y)
   implicit none
   real*8 :: t,x,y
   real*8, parameter :: a=,b=,c=,d= !os parâmetros devem ser
inseridos aqui
   real*8 fy
   fy = -c*y + d*x*y
   return
end function
```

3.3 e 3.4 - Evolução das populações, espaço de fase e ponto fixo

Em um programa FORTRAN, aproveitando do código do item anterior, foi possível calcular a evolução das populações de presa e predador com respeito ao tempo e traçar a sua trajetória y(t) X x(t) para diferentes valores de X0 e Y0 iniciais, utilizando os valores de a = 2/3, b = 4/3 e c = d = 1.

Para encontrar o(s) ponto(s) fixo(s), toma-se as equações diferenciais que dizem respeito à evolução das populações e encontra-se suas raízes, ou seja, os pontos em que elas são zero.

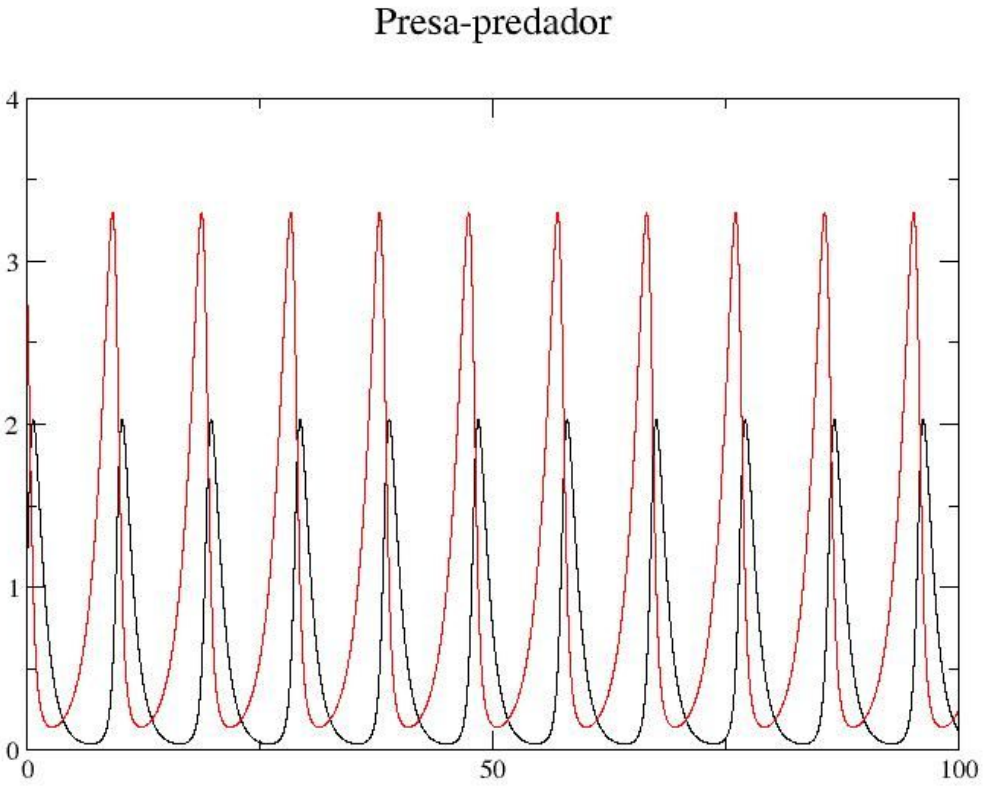
Resolvendo a equação diferencial para dx/dt = 0 e dy/dt =0, buscando os pontos fixos, encontra-se como ponto fixo o ponto ($\frac{c}{d}$, $\frac{a}{b}$),onde x = $\frac{c}{d}$ e y = $\frac{a}{b}$ e,

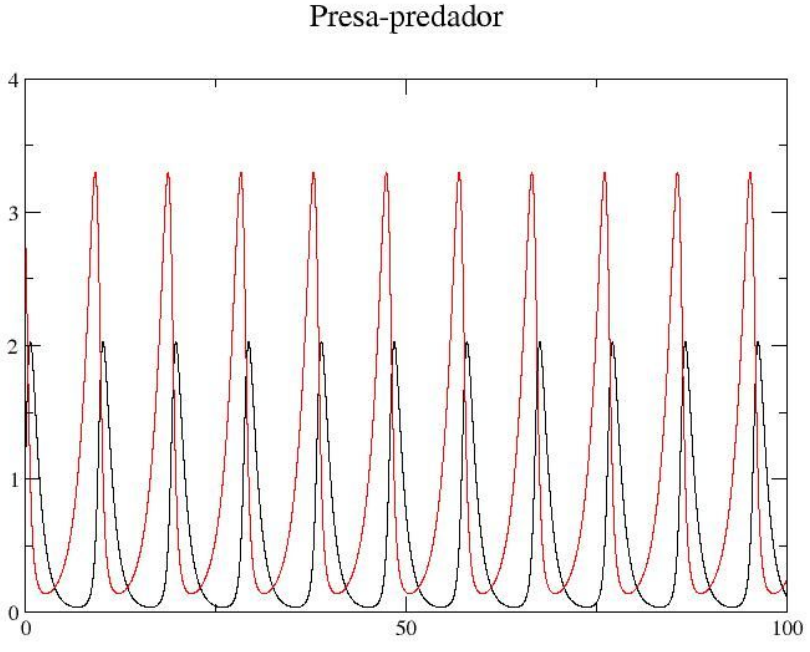
substituindo os valores dos parâmetros dados no problema, obtém-se: o ponto (1,½), onde x=1 e y=½. Assim, tomando esses pontos como iniciais deve-se notar que a população não variará com o tempo. Note que esse é um **ponto de equilíbrio estável** , por essa razão ao analisar-se a evolução da população com o tempo para valores próximos deste nota-se o formato mais claro de uma elipse no gráfico de x(t) X y(t).

Nas próximas páginas estão os cinco gráficos da evolução das populações de presa-**predador** com respeito ao tempo, onde a população em vermelho se refere às presas(X) e em preto aos predadores(Y), além da trajetória y(t) por x(t) da população.

Valores iniciais próximos mas não iguais ao ponto fixo apresentam trajetórias mais próximas de algo circular, enquanto valores mais distantes tendem se aproximar de uma reta, com valores nos cantos inferior e esquerdo do gráfico próximos a zero.

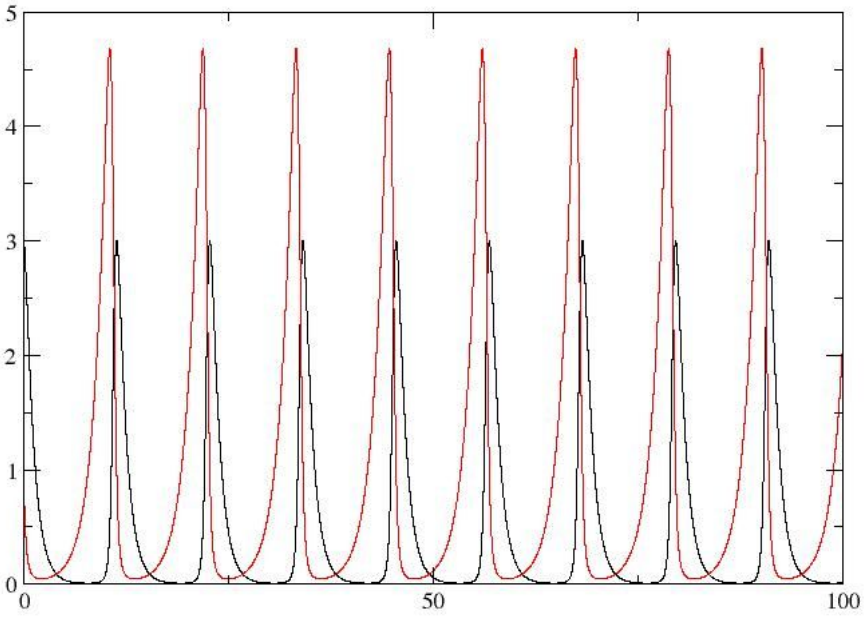
Primeiro caso: ponto inicial (3 , 1)



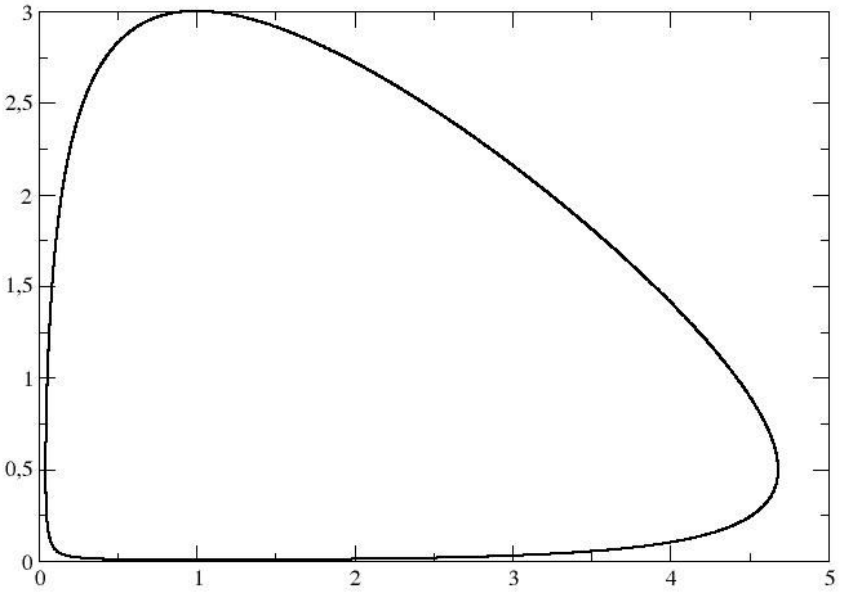


Segundo caso: ponto inicial (1 , 3)

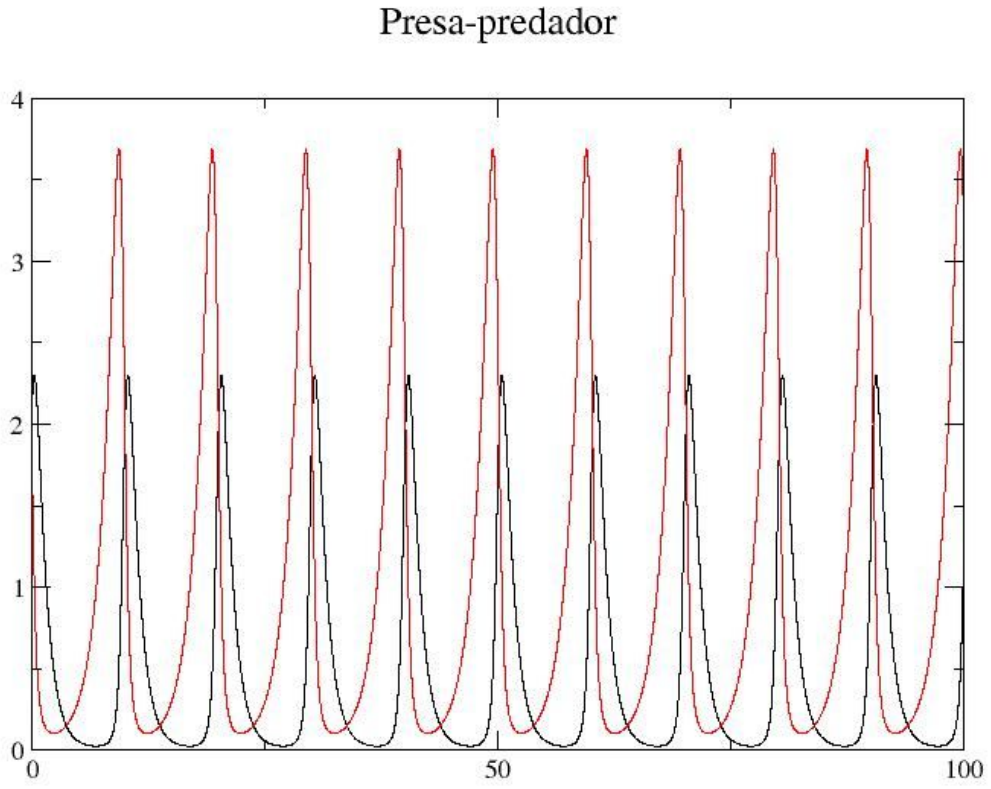
Presa-predador

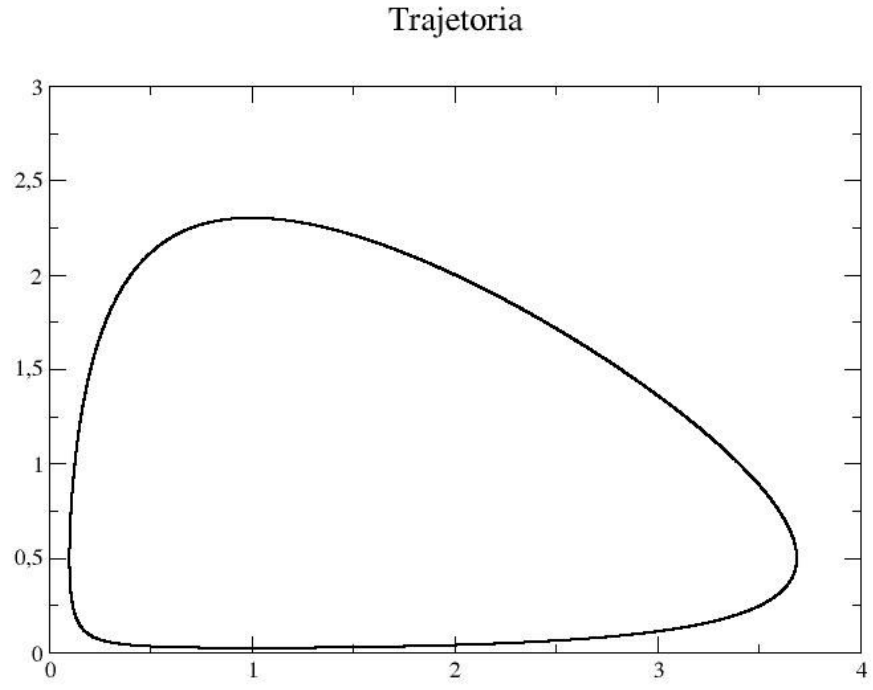


Trajetoria

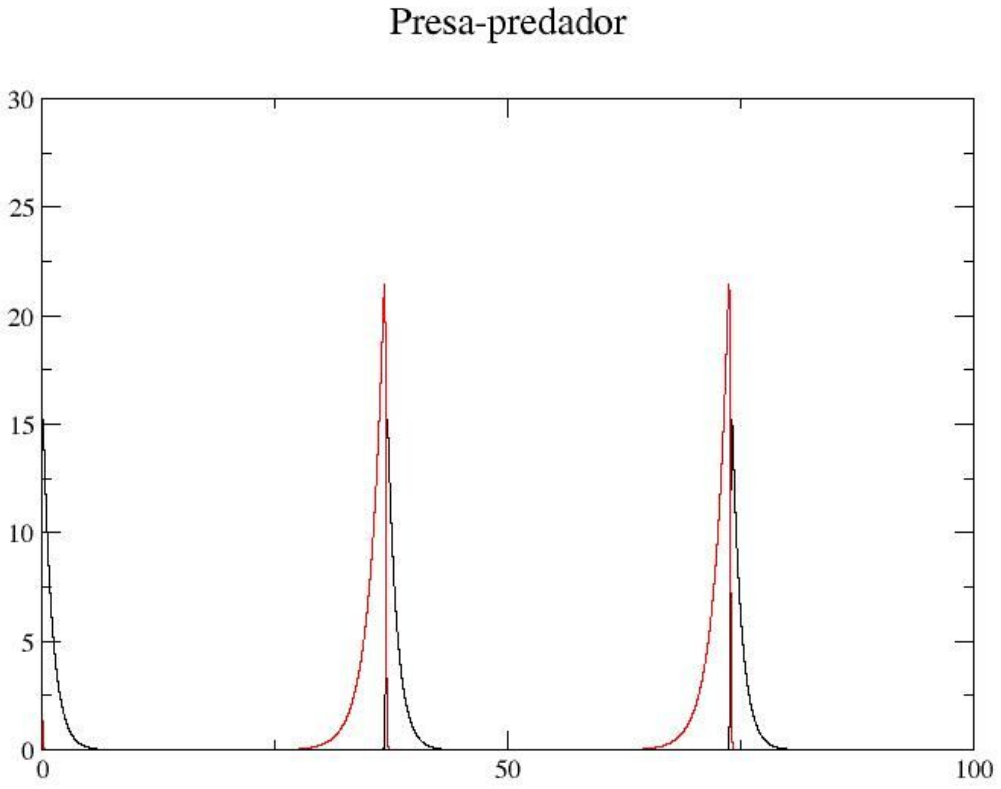


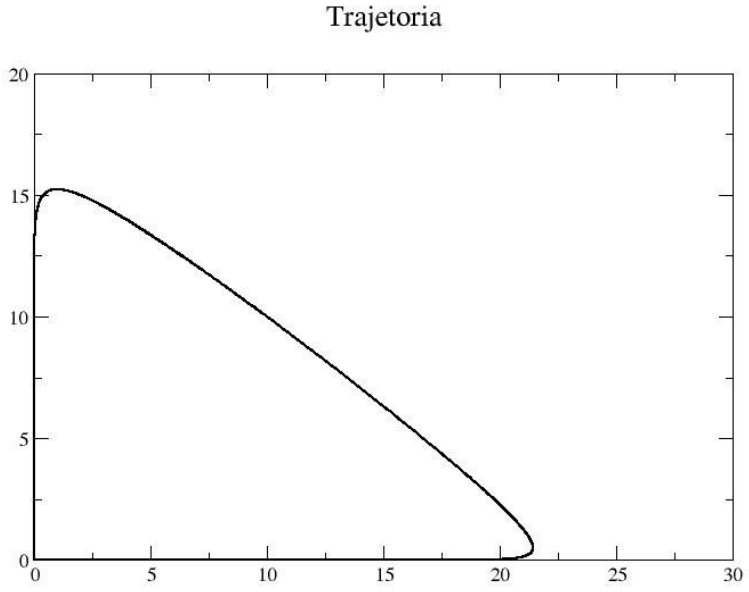
Terceiro caso: ponto inicial (2 , 2)



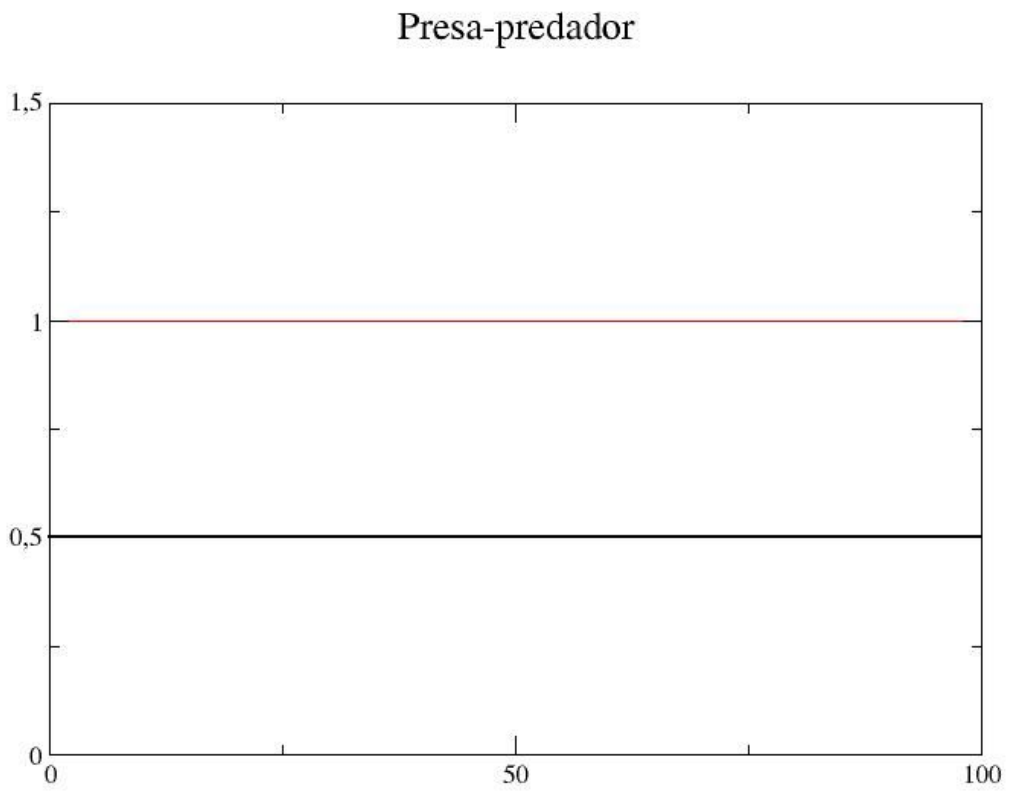


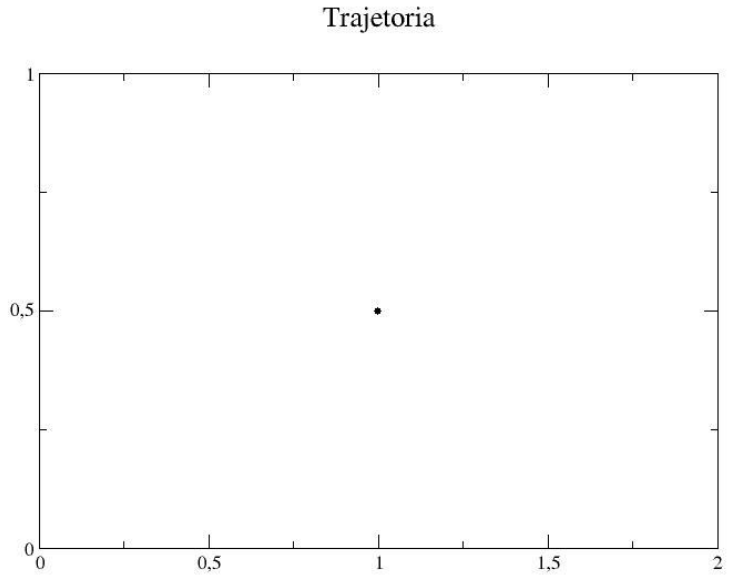
Quarto caso: ponto inicial (10 , 10)





Quinto caso: ponto inicial (1 , 0.5)






```
program predadorpresaRK4
```

```
!código runge-kutta para solução do problema de predador-presa

!dx/dt = ax - bxy (presas)
!dy/dt = -cy +dxy (predadores)

    implicit none

    real*8,external :: fx,fy
    real*8 :: x,y,vx,vy
    real*8 :: kx(5),ky(5)
    real*8 :: T,dT,Tf
    integer :: N,i,j,tempo
    real*8 :: iniciaisx(5),iniciaisy(5)
    character :: jc(1)


    iniciaisx(1) = 3.d0
    iniciaisy(1) = 1.d0

    iniciaisx(2) = 1.d0
    iniciaisy(2) = 3.d0

    iniciaisx(3) = 2.d0
    iniciaisy(3) = 2.d0

    iniciaisx(4) = 10.d0
    iniciaisy(4) = 10.d0

    iniciaisx(5) = 1.d0
    iniciaisy(5) = 0.5d0

    do j=1,5

     jc(1) = char(48+j)

        open(10,file="presa"//jc(1)//".dat")
        open(20,file="predador"//jc(1)//".dat")
        open(30,file="traj"//jc(1)//".dat")

     !condições iniciais

     x = iniciaisx(j)
     y = iniciaisy(j)

     !intervalo de iteração
```

```
      dT = 0.001d0
      Tf = 100.d0
      N = int(Tf/dT)
      T = 0.d0

      !loop

      do i=1,N

          !atualiza o tempo
          T = T + dT

          !atualiza as funções do tempo(X e Y)
          kx(1) = fx(x,y)
          ky(1) = fy(x,y)
          kx(2) = fx(x+0.5d0*dT*kx(1),y+0.5d0*dT*ky(1))
          ky(2) = fy(x+0.5d0*dT*kx(1),y+0.5d0*dT*ky(1))
          kx(3) = fx(x+0.5d0*dT*kx(2),y+0.5d0*dT*ky(2))
          ky(3) = fy(x+0.5d0*dT*kx(2),y+0.5d0*dT*ky(2))
          kx(4) = fx(x+dT*kx(3),y+dT*ky(3))
          ky(4) = fy(x+dT*kx(3),y+dT*ky(3))

          x = x + dT/6.d0*(kx(1) + 2*(kx(2)+kx(3)) + kx(4))
          y = y + dT/6.d0*(ky(1) + 2*(ky(2)+ky(3)) + ky(4))

          write(10,*)t,x
          write(20,*)t,y
          write(30,*)x,y

      enddo

     close(10)
     close(20)
     close(30)

     enddo

end program


function fx(x,y)
   implicit none
   real*8 :: t,x,y
   real*8, parameter :: a=2.d0/3.d0,b=4.d0/3.d0,c=1.d0,d=1.d0
   real*8 fx
   fx = a*x - b*x*y
```

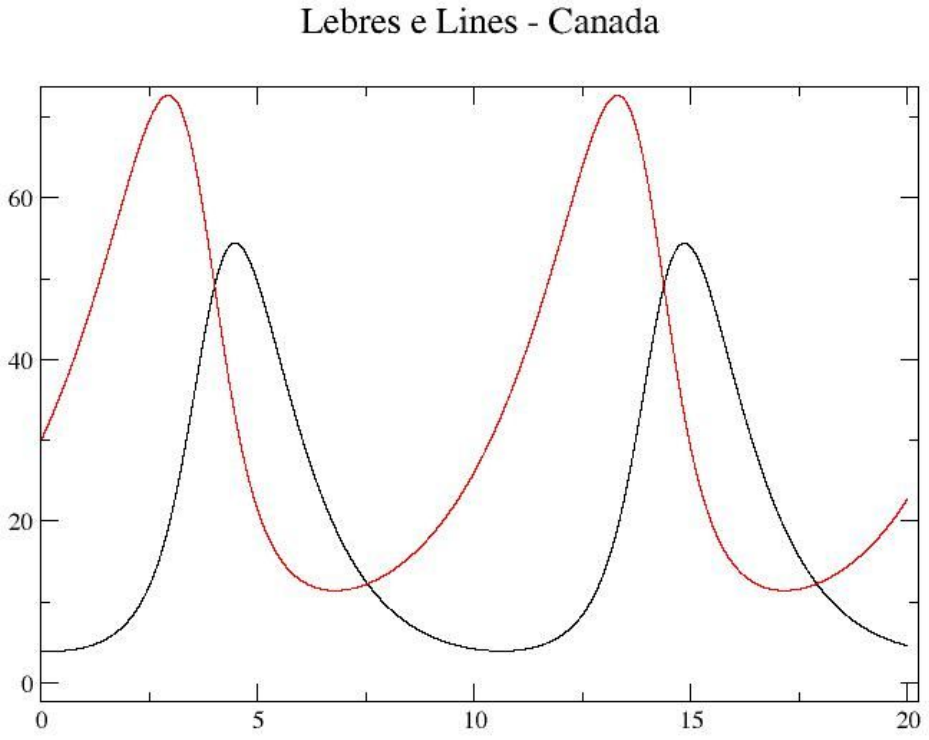
```
   return
end function

function fy(x,y)
   implicit none
   real*8 :: t,x,y
   real*8, parameter :: a=2.d0/3.d0,b=4.d0/3.d0,c=1.d0,d=1.d0
   real*8 fy
   fy = -c*y + d*x*y
   return
end function
```

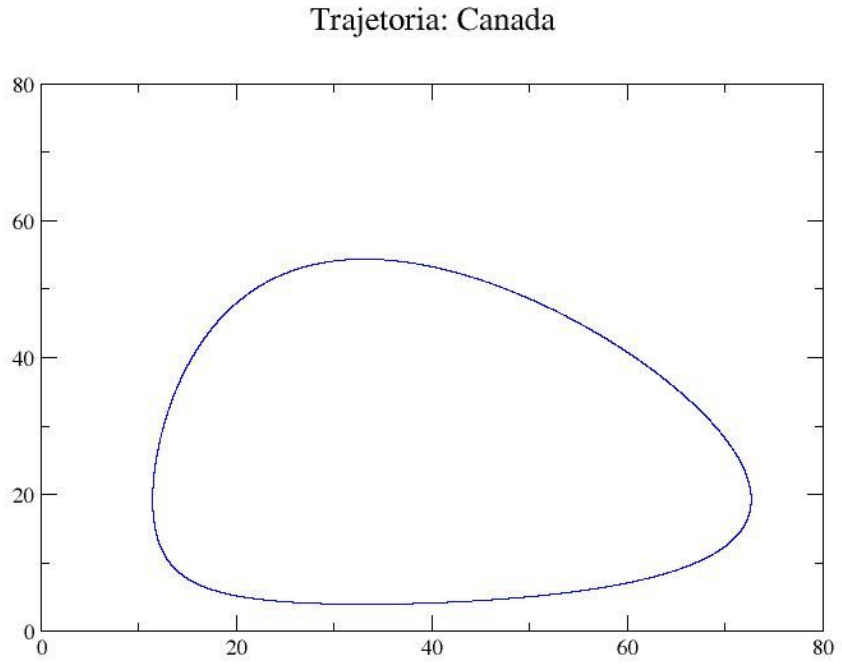
3.5 - Análise experimental: lebres e linces no Canadá

Tabela de progresso da população para efeitos de comparação com o resultado experimental:

1900	**30.000000000000000**	**4.0000000000000000**
1901	**43.842206119503459**	**4.4004051865013185**
1902	**61.637404384971781**	**7.5898076084670114**
1903	**72.587229849326519**	**20.602677047572751**
1904	**49.318962475066790**	**49.003237898286891**
1905	**21.403592646523748**	**49.516336656785853**
1906	**12.690559682036767**	**30.688512350707178**
1907	**11.505956345633884**	**16.874383759353709**
1908	**13.539438706990705**	**9.4246317202025516**
1909	**18.215041794437461**	**5.7797753047584717**
1910	**26.090027694610477**	**4.2168307310959339**
1911	**38.167773736707517**	**4.0580754978929425**
1912	**54.879956766782449**	**5.8530687327768769**
1913	**71.057300464734880**	**13.760053187391204**
1914	**62.052601205459233**	**38.732455393028410**
1915	**28.953526821092765**	**53.893154303397139**
1916	**14.549295646889069**	**37.565011189683048**
1917	**11.470939761294385**	**21.103032415460060**
1918	**12.487294599027429**	**11.601252212552630**
1919	**16.162389542766867**	**6.8147600938935922**

1920 22.753846941336398 4.6146546420553189






```
program predadorpresaRK4
```

```
!código runge-kutta para solução do problema de predador-presa

!dx/dt = ax - bxy (presas)
!dy/dt = -cy +dxy (predadores)

    implicit none

    real*8,external :: fx,fy
    real*8 :: x,y,vx,vy
    real*8 :: kx(5),ky(5)
    real*8 :: T,dT,Tf
    integer :: N,i,tempo

!open(10,file="presaXt.dat")
!open(20,file="predadorXt.dat")
open(30,file="trajetoria.dat")

!condições iniciais

x = 30.d0
y = 4.d0

!intervalo de iteração

dT = 0.001d0
Tf = 100.d0
N = int(Tf/dT)
T = 0.d0

!loop

do i=1,N

    !atualiza o tempo
    T = T + dT

    !atualiza as funções do tempo(X e Y)
    kx(1) = fx(x,y)
    ky(1) = fy(x,y)
    kx(2) = fx(x+0.5d0*dT*kx(1),y+0.5d0*dT*ky(1))
    ky(2) = fy(x+0.5d0*dT*kx(1),y+0.5d0*dT*ky(1))
    kx(3) = fx(x+0.5d0*dT*kx(2),y+0.5d0*dT*ky(2))
    ky(3) = fy(x+0.5d0*dT*kx(2),y+0.5d0*dT*ky(2))
    kx(4) = fx(x+dT*kx(3),y+dT*ky(3))
    ky(4) = fy(x+dT*kx(3),y+dT*ky(3))
```

```
    x = x + dT/6.d0*(kx(1) + 2*(kx(2)+kx(3)) + kx(4))
    y = y + dT/6.d0*(ky(1) + 2*(ky(2)+ky(3)) + ky(4))

    write(10,*)t,x
    write(20,*)t,y
    write(30,*)x,y

enddo

end program


function fx(x,y)
   implicit none
   real*8 :: t,x,y
   !real*8, parameter :: a=2.d0/3.d0,b=4.d0/3.d0,c=1.d0,d=1.d0
   real*8, parameter :: a=0.481d0,b=0.025d0,c=0.927d0,d=0.028d0
   real*8 fx
   fx = a*x - b*x*y
   return
end function

function fy(x,y)
   implicit none
   real*8 :: t,x,y
   !real*8, parameter :: a=2.d0/3.d0,b=4.d0/3.d0,c=1.d0,d=1.d0
   real*8, parameter :: a=0.481d0,b=0.025d0,c=0.927d0,d=0.028d0
   real*8 fy
   fy = -c*y + d*x*y
   return
end function
```