Universidad Nacional de El Salvador

Facultad Multidiscplinaria de Occidente

Departamento de Ingenieria y Arquitectura

Asignatura: Analisis Numerco.

Primer Parcial evaluado

Catedratico: Ing. Xenia Ivette Godoy

Estudiante: Meda Margueiz, Christian Eduardo MM17017

Fecha de entrega: Miercoles, 24 de marzo de 2,020

Ejercicio1

$$f(x) = \text{sen}(x) - 6x - 5$$

i)Algoritmo
Algoritmo por biseccion

```
from sympy import *

def f(num):
      return (sin(num)-6*num-5)

x1=-2
x2=-0.9

errorAproximado=1
errorEspecifico=0.05
iterador=1
condicion=""
anterior=0
actual=0

if (f(x1)*f(x2)<0):
      print "Valores Validos"
      print "i | x1 | x2 | xr | fx1 | fx2 | fxr | fx1*fxr | condicion | EA"
      xr=(x1+x2)/2
      if f(x1)*f(xr)<0:
            condicion="<"
      elif f(x1)*f(xr)>0:
            condicion=">"

      print iterador," | ",x1," | ",x2," | ",xr," | ",f(x1)," | ",f(x2)," |
",f(xr)," | ",f(x1)*f(xr)," | ",condicion," | ",errorAproximado

      if f(x1)*f(xr)<0:
            x2=xr
            condicion="<"
      elif f(x1)*f(xr)>0:
            condicion=">"
            x1=xr
      else:
            print "La raiz es: ",xr
            errorAproximado=0

      while errorAproximado>errorEspecifico:
            anterior=xr
            xr=(x1+x2)/2
            actual=xr
            fx1=f(x1)
            fx2=f(x2)
            fxr=f(xr)
            if f(x1)*f(xr)<0:
                  x2=xr
                  condicion="<"
            elif f(x1)*f(xr)>0:
                  condicion=">"
                  x1=xr
            else:
```

```
                print "La raiz es: ",xr
                break

          if (iterador!=1):
                errorAproximado=Abs((anterior-actual)/actual)*100

          print iterador," | ",x1," | ",x2," | ",xr," | ",fx1," | ",fx2,"
| ",fxr," | ",fx1*fxr," | ",condicion," | ",errorAproximado
          iterador=iterador+1
else:
     print "Valores invalidos"

print "La raiz es: ",xr
print "EA: ",errorAproximado
```

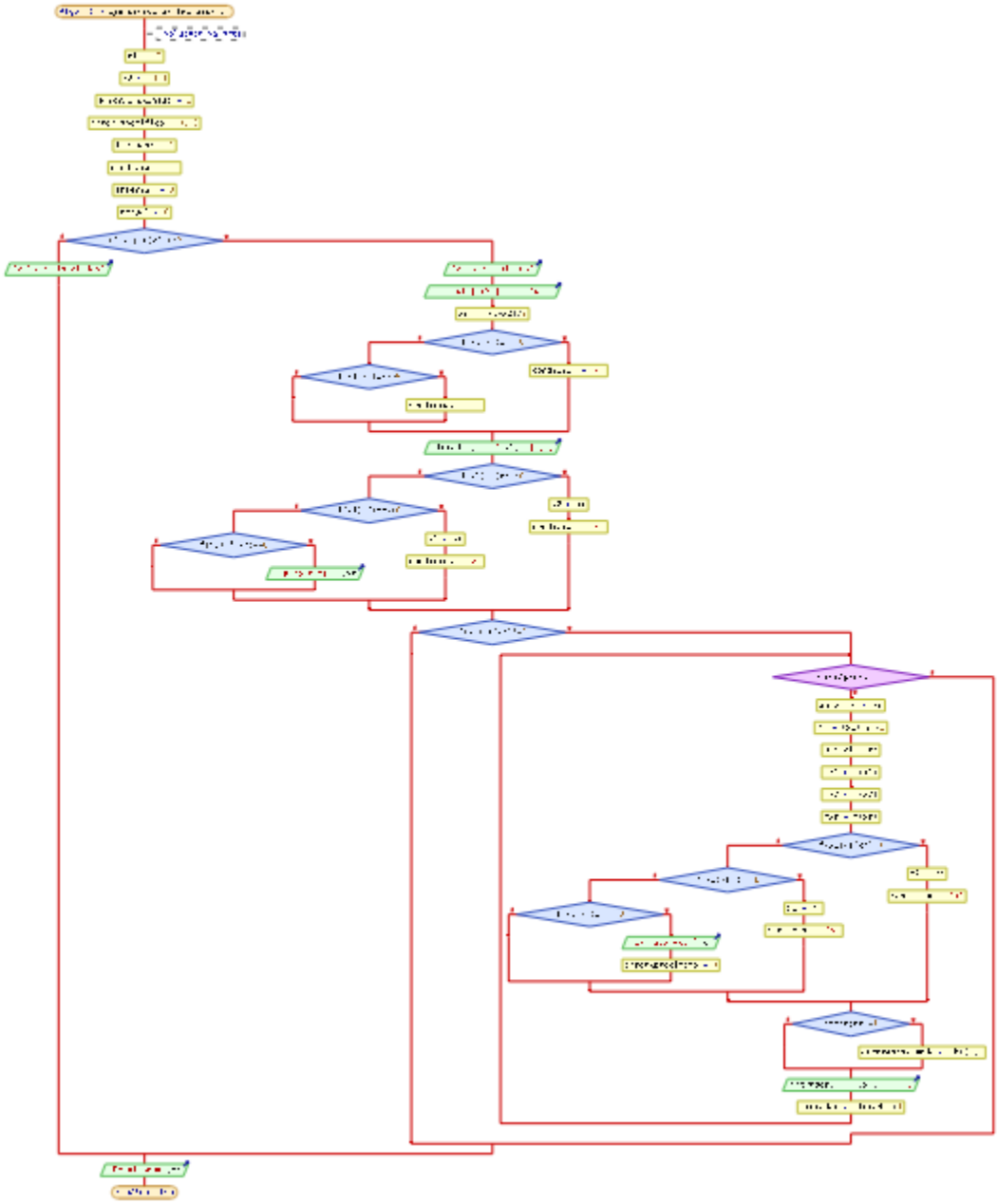
Algoritmo por newton

```
from sympy import *

def f(num):
     return float(sin(num)-6*num-5)

def dx(num):
     return float(cos(num)-6)

def dxx(num):
     return float(-sin(num))

def newton(x):
    return float(x-(f(x)/dx(x)))

def solucionByNewton(x):
     convergencia=(f(x)*dxx(x))/pow(dx(x),2)

     if convergencia<1:
          errorAproximado=1
          errorEspecifico=0.05
          iterador=1
          anterior=0
          actual=0

          fx=f(x)
          fprima=dx(x)
          xi=newton(x)
          errorAproximado=Abs((x-xi)/xi)*100
          print "iterador | f(xi) | dx(xi) | xi+1 | Ea"
          print iterador," | ",fx," | ",fprima," | ",xi," |
",errorAproximado

          while errorAproximado>errorEspecifico:
                x=xi
                fx=f(x)
                fprima=dx(x)
                xi=newton(x)
                errorAproximado=Abs((x-xi)/xi)*100
                print iterador," | ",fx," | ",fprima," | ",xi," |
",errorAproximado
                iterador=iterador+1
     print "El valor de la raiz es: ",xi
     print "Con el error de: ",errorAproximado
```

solucionByNewton(-2)

ii)Flujograma

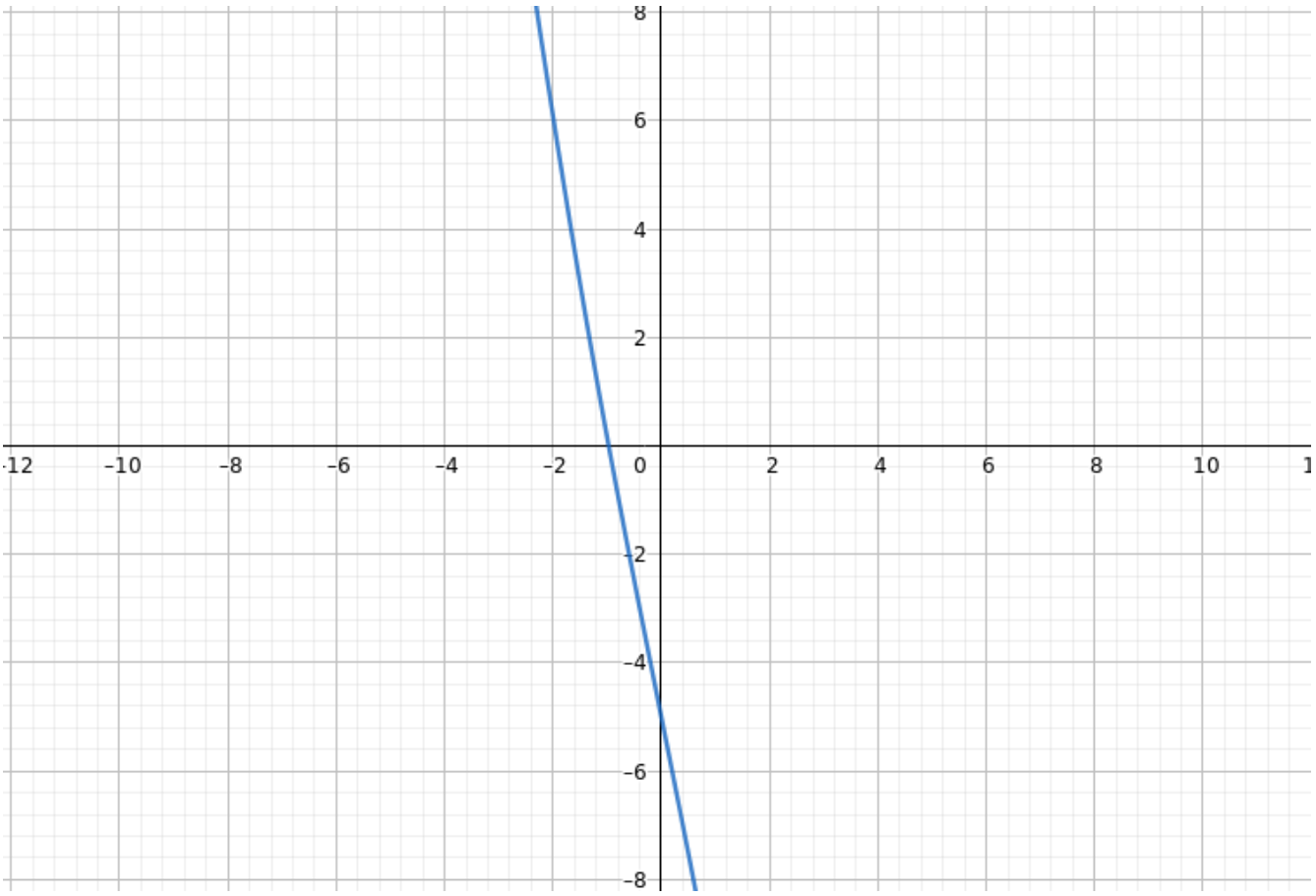
Por biseccion:



Por newton rapson

```
Algoritmo Ejercicio1 parte2 newton
    x ← -2
    errorAproximado ← 1
    errorEspecifico ← 0.05
    iterador ← 1
    anterior ← 0
    actual ← 0
    fx ← f(x)
    fprima ← derivada(x)
    xi ← newton(x)
    errorAproximado ← ab(...
    Escribir 'iterador | f(xi) | dx...
    Escribir iterador,' | ',fx,' | ...
    Mientras errorAproxim...
        x ← xi
        fx ← f(x)
        fprima ← derivada(x)
        xi ← newton(x)
        errorAproximado ← ab(...
        Escribir iterador,' | ',fx,' | ...
        iterador ← iterador+1
    Escribir 'El valor de la raiz e...
    Escribir 'Con el error de: ',er...
FinAlgoritmo
```

iii)Imagen de la grafica



iv)Determinar valores iniciales
Para la biseccion
x1=-2
x2=-0.9
Para newton
x=-2
v)Tabla de iteraciones

Por biseccion:

1	-2	-0.9	-1.45	6.0907025732	-0.3833269096	2.707287009	0	> 1
1	-1.175	-0.9	-1.175	2.707287009	-0.3833269096	1.1273101613	3.0519521548	> 1
2	-1.0375	-0.9	-1.0375	1.1273101613	-0.3833269096	0.3638640171	0.4101876038	> 13.2530120482
3	-1.0375	-0.96875	-0.96875	0.3638640171	-0.3833269096	-0.0116784447	-0.0042493658	< 7.0967741935
4	-1.003125	-0.96875	-1.003125	0.3638640171	-0.0116784447	0.175594682	0.0638925864	> 3.4267912773
5	-0.9859375	-0.96875	-0.9859375	0.175594682	-0.0116784447	0.0818349667	0.0143697849	> 1.7432646593
6	-0.97734375	-0.96875	-0.97734375	0.0818349667	-0.0116784447	0.0350476488	0.0028681232	> 0.8792965627
7	-0.973046875	-0.96875	-0.973046875	0.0350476488	-0.0116784447	0.0116769712	0.0004092504	> 0.441589723
8	-0.973046875	-0.9708984375	-0.9708984375	0.0116769712	-0.0116784447	-0.0000026416	-0.0000000308	< 0.221283444
9	-0.9719726562	-0.9708984375	-0.9719726562	0.0116769712	-0.0000026416	0.0058366882	0.0000681548	> 0.1105194414
10	-0.9714355469	-0.9708984375	-0.9714355469	0.0058366882	-0.0000026416	0.0029169042	0.0000170251	> 0.0552902739
11	-0.9711669922	-0.9708984375	-0.9711669922	0.0029169042	-0.0000026416	0.0014571015	0.0000042502	> 0.0276527816

Resultado= 0.9711669922

Por newton:

iterador	f(xi)	dx(xi)	xi+1	Ea
1	6.0907025732	-6.4161468365	-1.0507226957	90.3451793931
1	0.4365535825	-5.503055965	-0.9713933948	8.1665472848
2	0.0026877709	-5.4358504087	-0.9708989421	0.050927308
3	0.0000001009	-5.4354422216	-0.9708989235	0.0000019124

vi)
Respuestas:

Por la biseccion
Resultado=-0.9711669922
Error aproximado= 0.027652781

Por newton rapson
La raiz es: -0.971166992188
EA: 0.0276527816184396

Ejercicio2

$$f(x) = -x^4 - 9x^3 - 5x^2 - 26x + 24$$

i)Algoritmo

Por Ferrari:

```
from sympy import *

print "Raices de -9X^4+25X^3-5x^2-26x+24"

a=-25/9
b=5/9
c=26/9
d=-24/9

P=(8*b-3*(pow(a,2)))/8
Q=(8*c-4*a*b+pow(a,3))/8
R=(256*d-64*a*c+16*pow(a,2)*b-3*pow(a,4))/256
print "2-variables para formar la cubica"

a2=-(P/2)
b2=(R)
c2=((4*P*R)-pow(Q,2))/8

print "3- Resolver por tartaglia"

p2=b2-pow(a2,2)/3
q2=c2-((a2*b2)/3)+2*pow(a,3)/27

D=(sqrt((4*pow(p2,3)+27*pow(q2,2))/108))
U=0

if (D>0):
      A=-q2/2+sqrt(pow(q2,2)/4+pow(q2,3)/27)
      B=-q2/2-sqrt(pow(q2,2)/4+pow(q2,3)/27)
      U=pow(A,(1/3))+pow(B,(1/3))
elif (D<0):
      cosfi=(((27)^0.5)*q2)/((2*p2)*(-p2)^0.5)
      fi=acos(cosfi)
      U=(2*(-p/3)^0.5)*(cos(fi/3))
else:
      U=2*((-q2/2)^(1/3))

V=sqrt(2*U-P)
W=Q/(-2*V)

x1=complex((V/2)-a/4,+sqrt(pow(V,2)-4*(U-W))/2)
x2=complex((V/2)-a/4,-sqrt(pow(V,2)-4*(U-W))/2)
x3=complex(-(V/2)-a/4,+sqrt(pow(V,2)-4*(U-W))/2)
x4=complex(-(V/2)-a/4,-sqrt(pow(V,2)-4*(U-W))/2)

print "Raices "
print x1
print x2
print x3
print x4
```

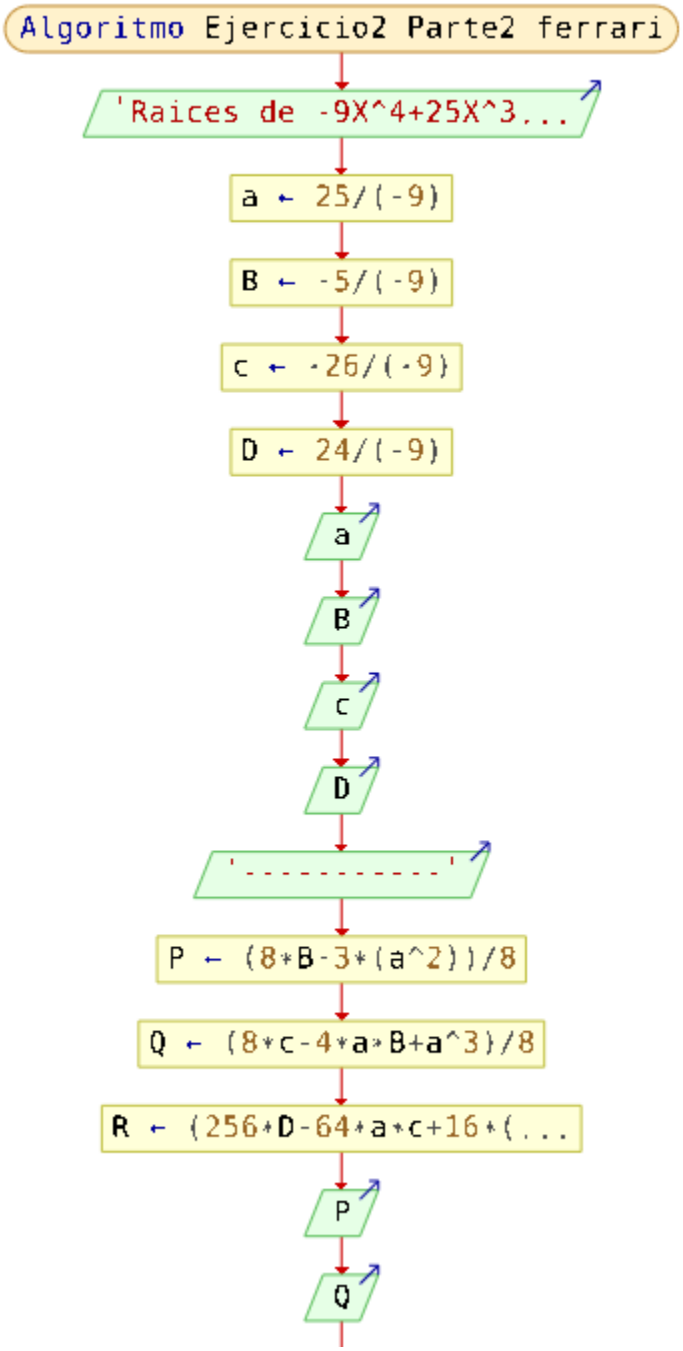
Por Bairstow

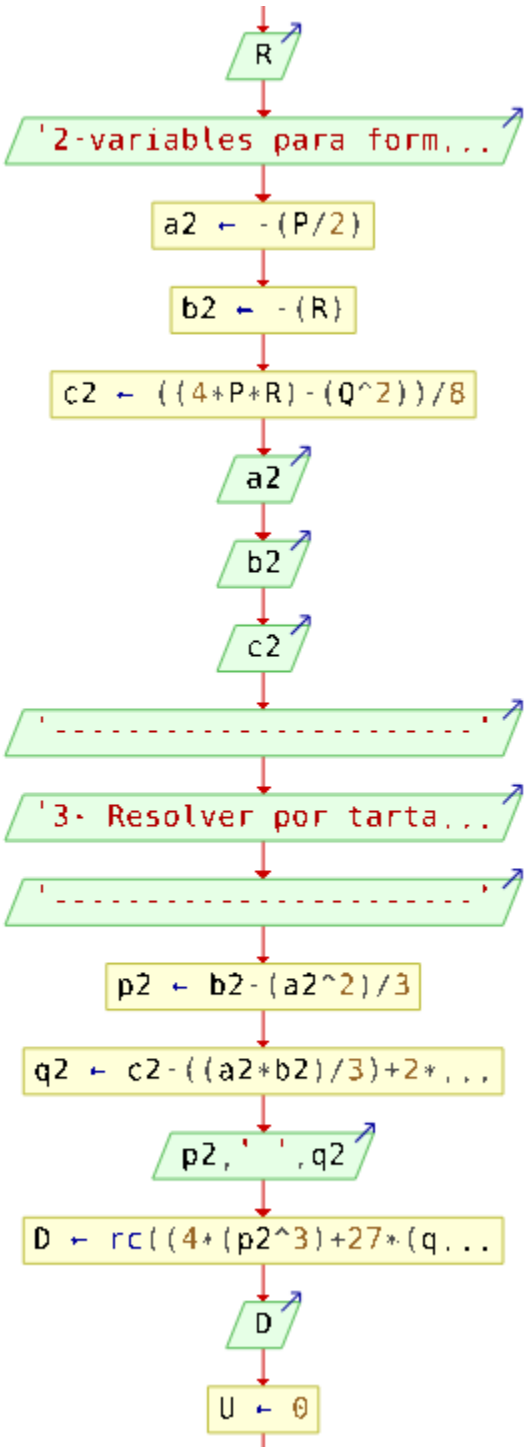
```
from sympy import *


r0=1.5
s0=1.5
Ear=0
Eas=0
a4=1
a3=-25/9
a2=5/9
a1=26/9
a0=-24/9
iterador=1
errorEspecifico=0.05


b4=float(a4)
b3=float(a3+b4*r0)
b2=float(a2+b3*r0+b4*s0)
b1=float(a1+b2*r0+b3*s0)
b0=float(a0+b1*r0+b2*s0)

c4=float(b4)
c3=0.222222222
c2=float(b2+c3*r0+c4*s0)
c1=float(b1+c2*r0+c3*s0)

print c3

deltaR=(-c3*b0+c2*b1)/(c1*c3-pow(c2,2))
deltaS=(-b1-(c2*deltaR))/c3

r=r0+deltaR
s=s0+deltaS

Ear=Abs(deltaR/r)*100
Eas=Abs(deltaS/s)*100

print iterador," | ",b1," | ",b0," | ",c3," | ",c2," | ",c1," | ",deltaR,"
| ",deltaS," | ",r," | ",s," | ",Ear," | ",Eas

while Ear>errorEspecifico or Eas>errorEspecifico:
      r0=r
      s0=s
      b4=a4
      b3=a3+b4*r0
      b2=a2+b3*r0+b4*s0
      b1=a1+b2*r0+b3*s0
      b0=a0+b1*r0+b2*s0

      c4=b4
      c3=b3+c4*r0
      c2=b2+c3*r0+c4*s0
      c1=b1+c2*r0+c3*s0

      deltaR=(-c3*b0+c2*b1)/(c1*c3-pow(c2,2))
      deltaS=(-b1-(c2*deltaR))/c3

      r=r0+deltaR
```

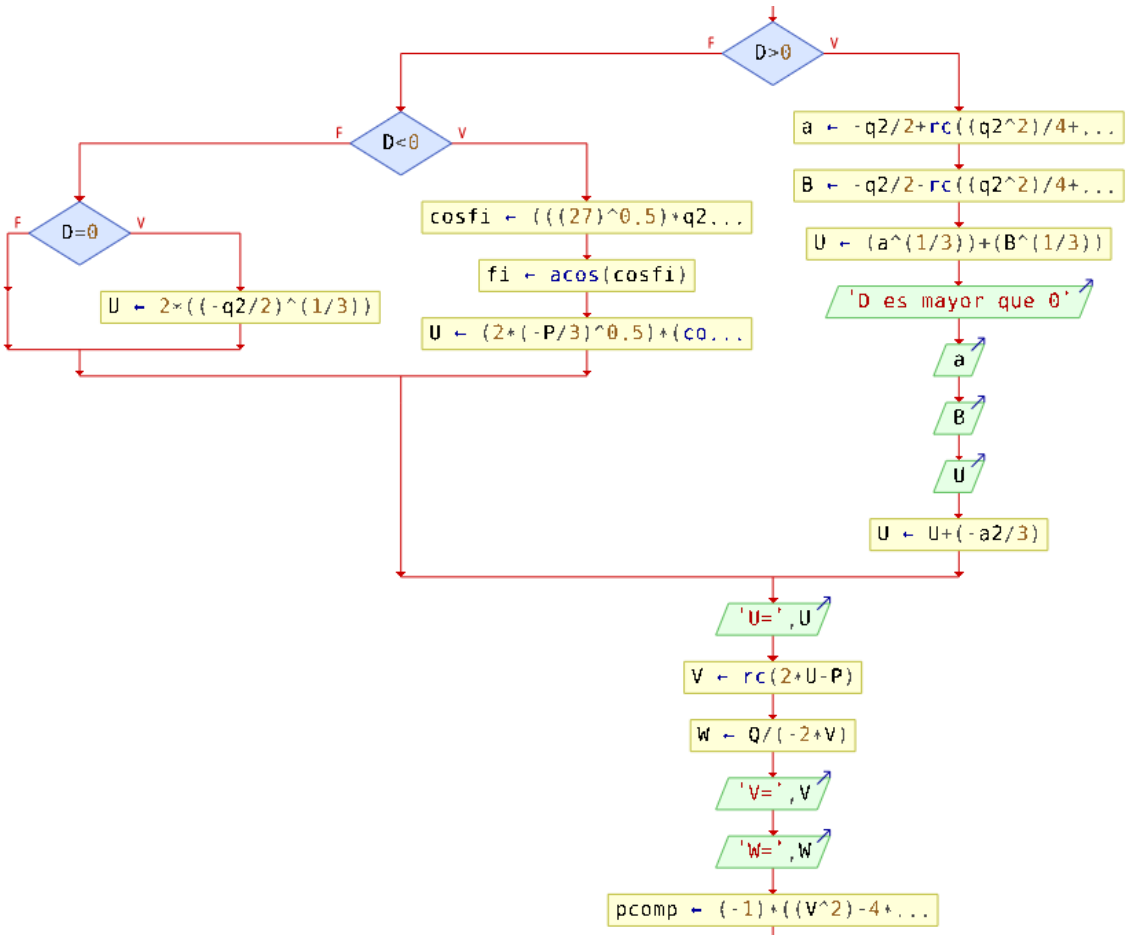
```
      s=s0+deltaS

      Ear=Abs(deltaR/r)*100
      Eas=Abs(deltaS/s)*100
      print iterador," | ",b1," | ",b0," | ",c3," | ",c2," | ",c1," |
",deltaR," | ",deltaS," | ",r," | ",s," | ",Ear," | ",Eas


x1=(r+sqrt(pow(r,2)+4*s))/2
x2=(r-sqrt(pow(r,2)+4*s))/2

b4=a4
b3=a3+b4*x1
b2=a2+b3*x1
b1=a1+b2*x1
b0=a0+b1*x1

c4=b4
c3=b3+c4*x2
c2=b2+c3*x2
c1=b1+c2*x2
d=(-1)*(pow(c3,2)-4*c4*c2)
x3=-c3/2
x4=-c3/2
print "Raices"
print x1
print x2
print x3,"+",d,"i/2"
print x4,"-",d,"i/2"
```

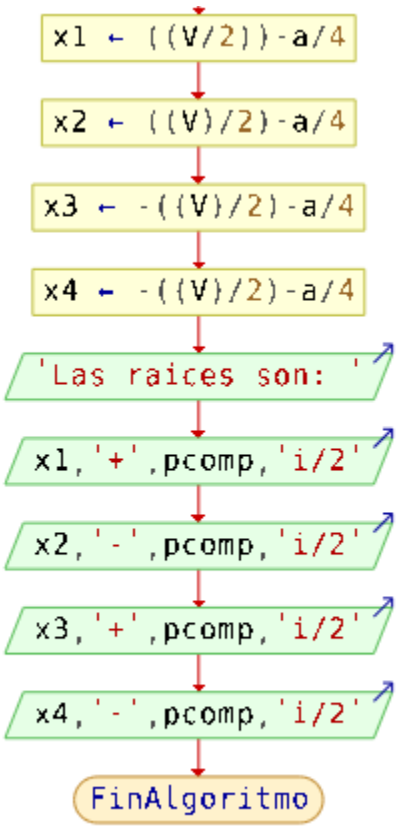
ii)Flujogramas

Por Ferrari

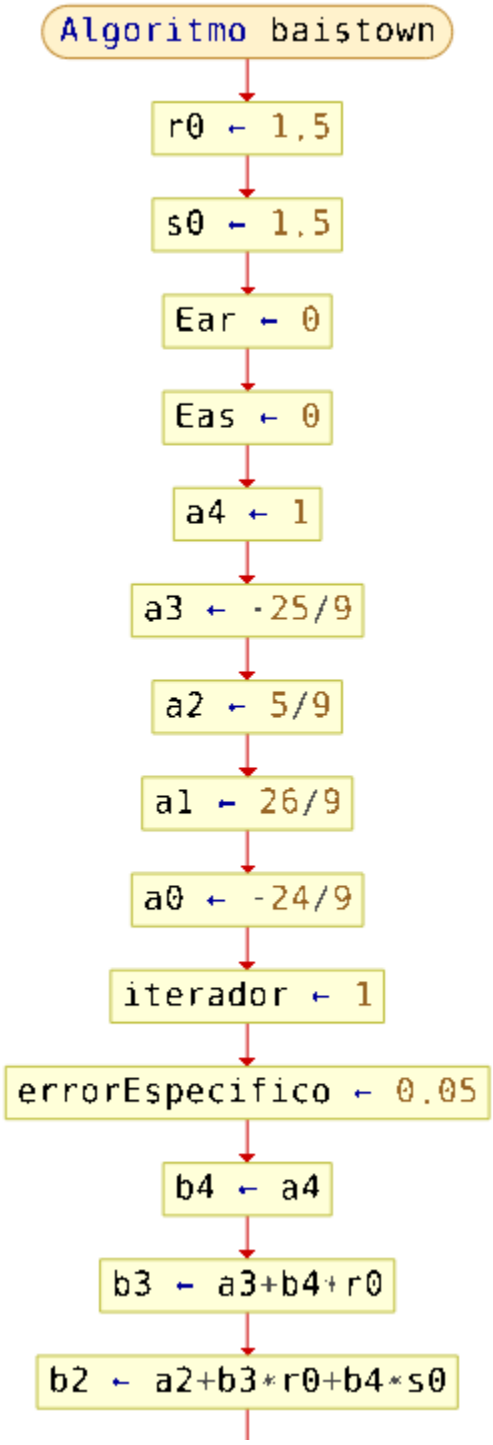


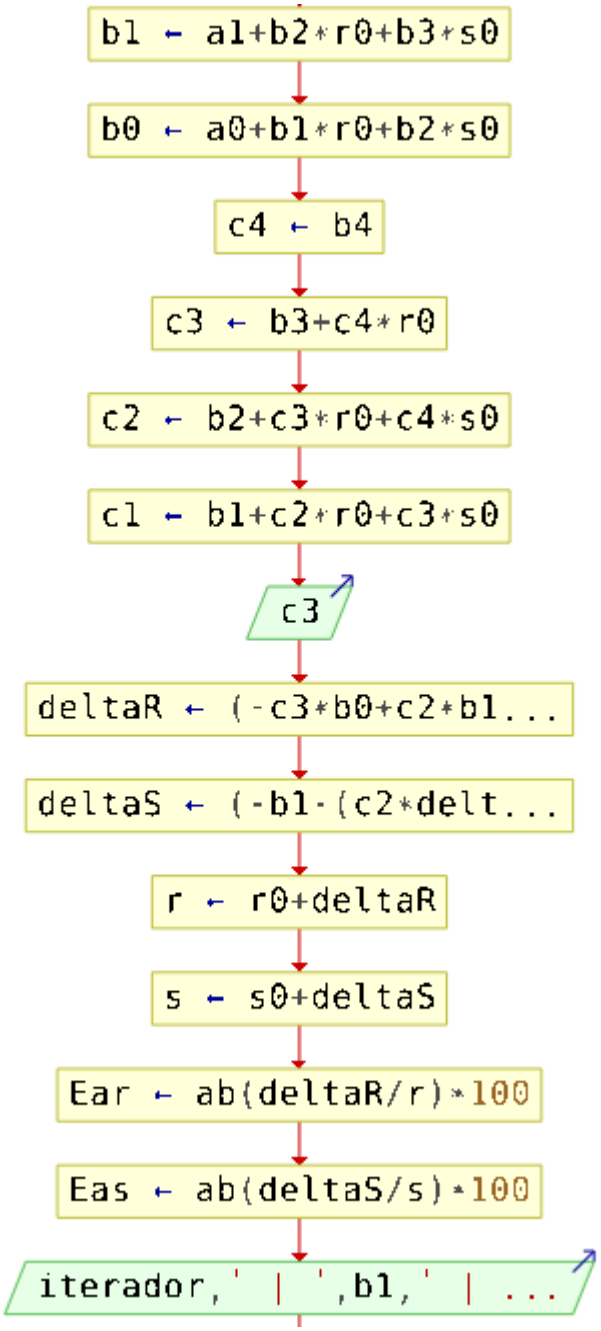

R
'2-variables para form...
a2 ← -(P/2)
b2 ← -(R)
c2 ← ((4*P*R)-(Q^2))/8
a2
b2
c2
'----------'
'3- Resolver por tarta...
'----------'
p2 ← b2-(a2^2)/3
q2 ← c2-((a2*b2)/3)+2*...
p2,' ',q2
D ← rc((4*(p2^3)+27*(q...
D
U ← 0

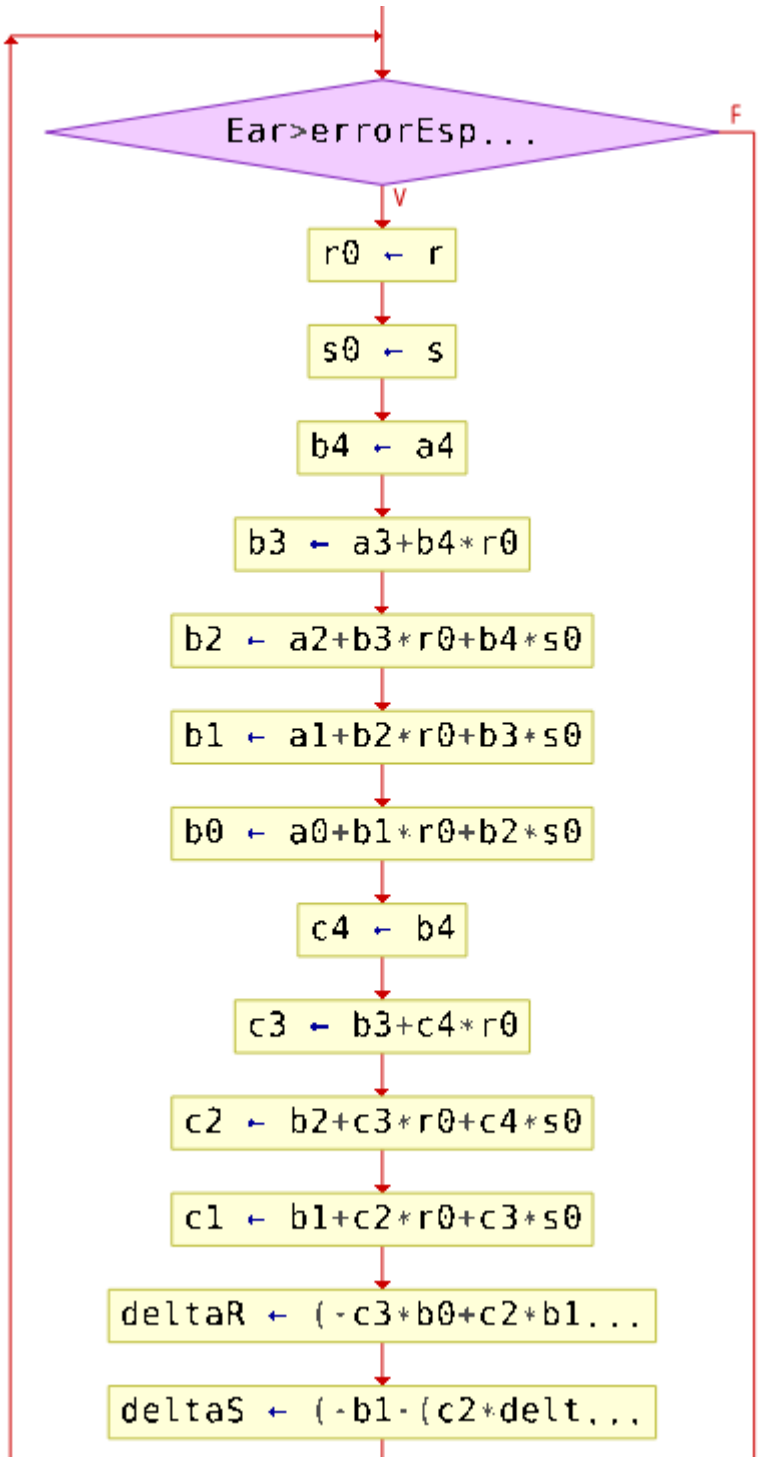

D>0
F
V
D<0
F
V
D=0
F
V
U ← 2*((-q2/2)^(1/3))
cosfi ← (((27)^0.5)*q2...
fi ← acos(cosfi)
U ← (2*(-P/3)^0.5)*(co...
a ← -q2/2+rc((q2^2)/4+...
B ← -q2/2-rc((q2^2)/4+...
U ← (a^(1/3))+(B^(1/3))
'D es mayor que 0'
a
B
U
U ← U+(-a2/3)
'U=',U
V ← rc(2*U-P)
W ← Q/(-2*V)
'V=',V
'W=',W
pcomp ← (-1)*((V^2)-4*...

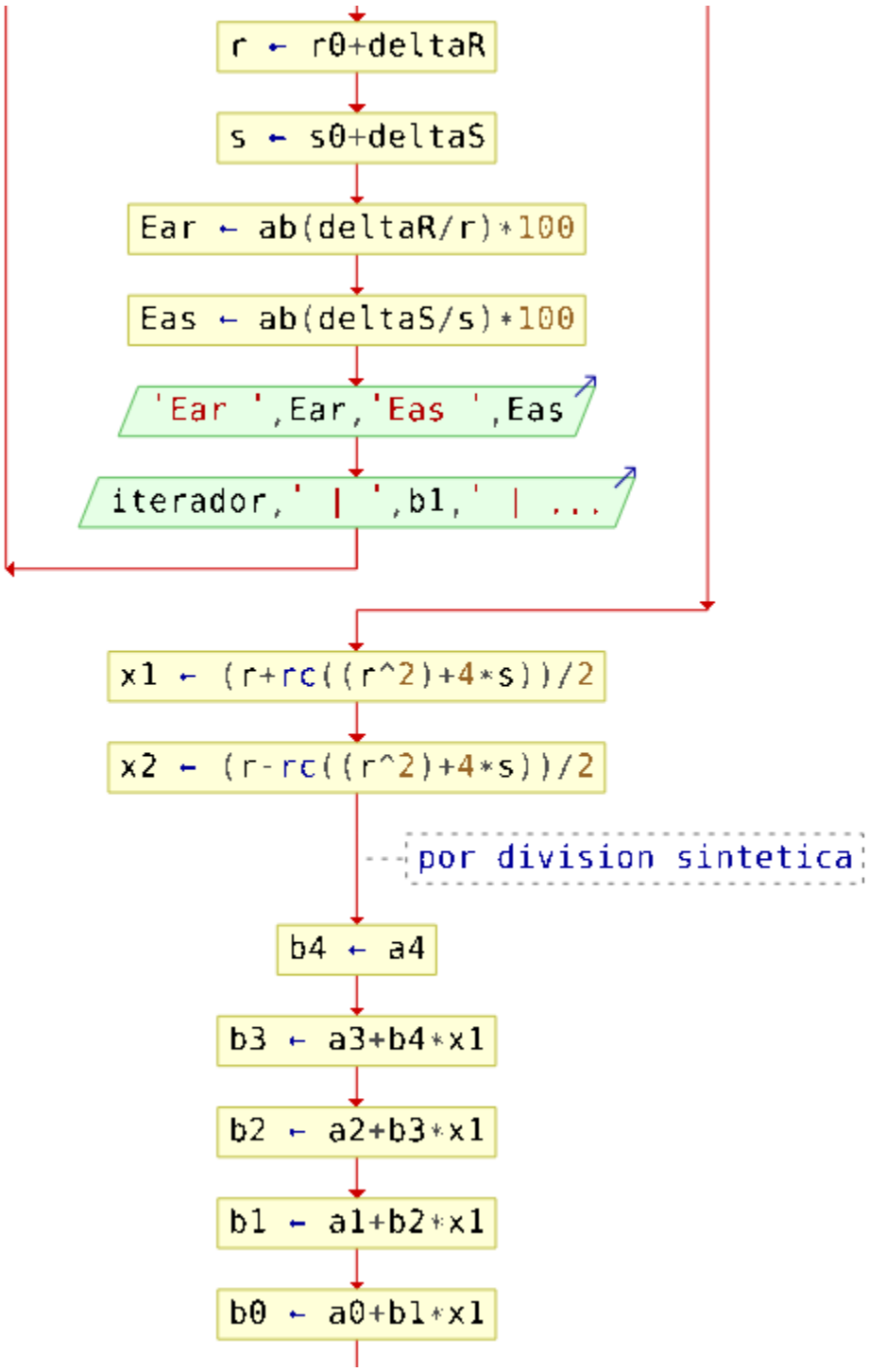

x1 ← ((V/2))-a/4
x2 ← ((V)/2)-a/4
x3 ← -((V)/2)-a/4
x4 ← -((V)/2)-a/4
'Las raices son: '
x1,'+',pcomp,'i/2'
x2,'-',pcomp,'i/2'
x3,'+',pcomp,'i/2'
x4,'-',pcomp,'i/2'
FinAlgoritmo

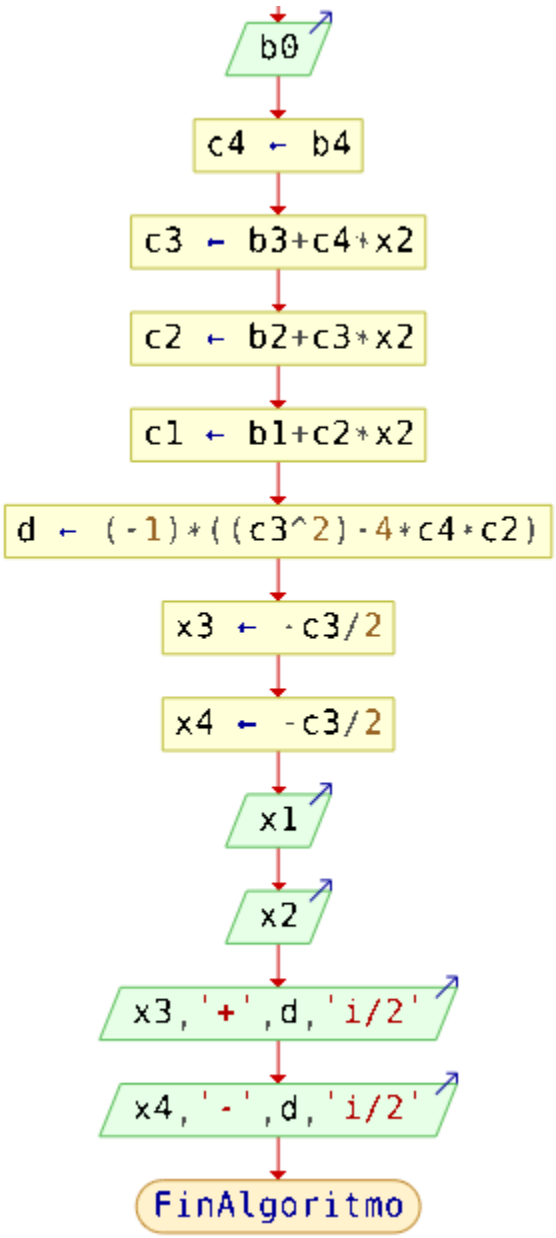
Por Bairstow:



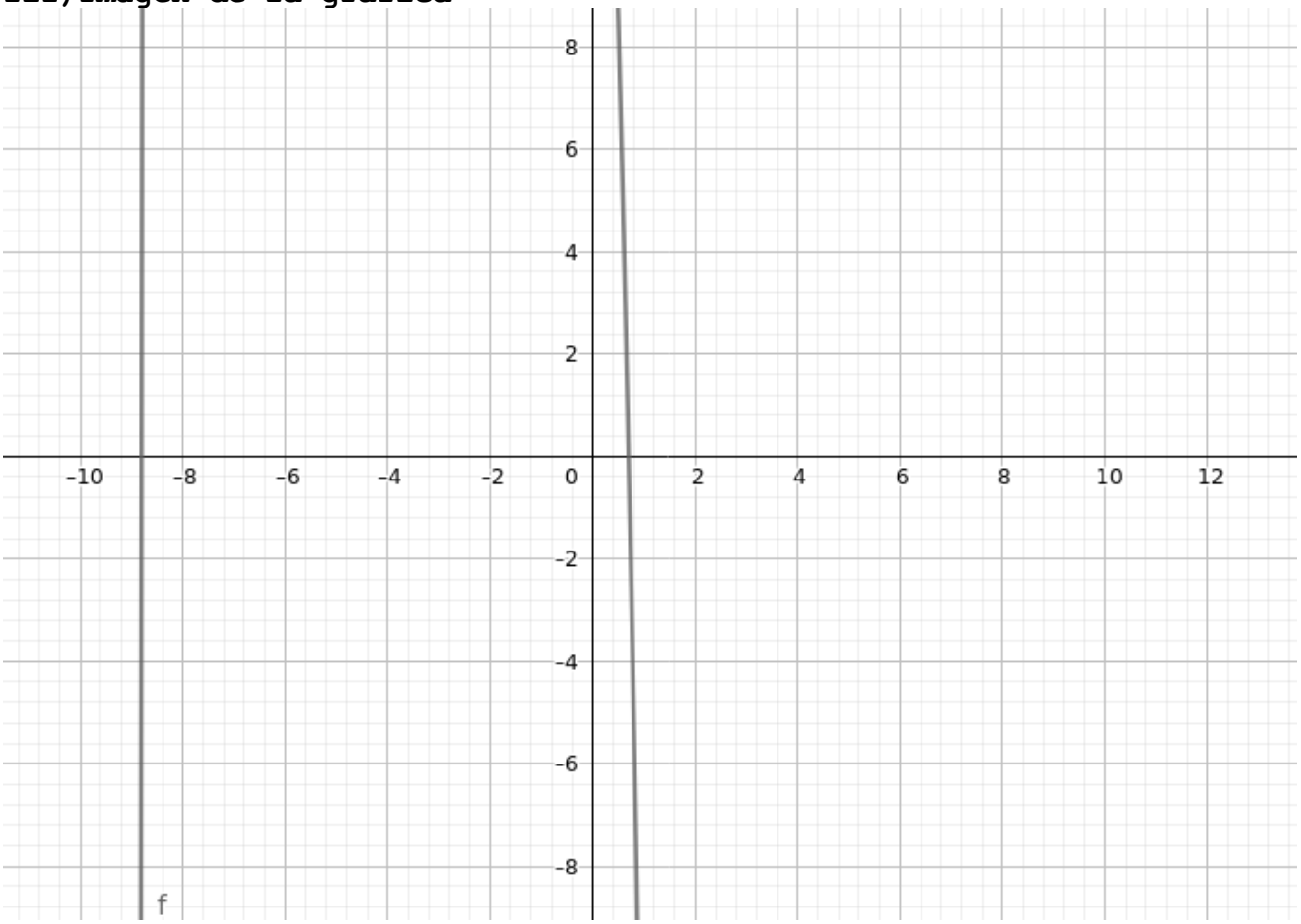

b1 ← a1+b2*r0+b3*s0
b0 ← a0+b1*r0+b2*s0
c4 ← b4
c3 ← b3+c4*r0
c2 ← b2+c3*r0+c4*s0
c1 ← b1+c2*r0+c3*s0
c3
deltaR ← (-c3*b0+c2*b1...
deltaS ← (-b1-(c2*delt...
r ← r0+deltaR
s ← s0+deltaS
Ear ← ab(deltaR/r)*100
Eas ← ab(deltaS/s)*100
iterador,' | ',b1,' | ...


Ear>errorEsp...
F
V
r0 ← r
s0 ← s
b4 ← a4
b3 ← a3+b4*r0
b2 ← a2+b3*r0+b4*s0
b1 ← a1+b2*r0+b3*s0
b0 ← a0+b1*r0+b2*s0
c4 ← b4
c3 ← b3+c4*r0
c2 ← b2+c3*r0+c4*s0
c1 ← b1+c2*r0+c3*s0
deltaR ← (-c3*b0+c2*b1...
deltaS ← (-b1-(c2*delt...


r ← r0+deltaR
s ← s0+deltaS
Ear ← ab(deltaR/r)*100
Eas ← ab(deltaS/s)*100
'Ear ',Ear,'Eas ',Eas
iterador,' | ',b1,' | ...
x1 ← (r+rc((r^2)+4*s))/2
x2 ← (r-rc((r^2)+4*s))/2
por division sintetica
b4 ← a4
b3 ← a3+b4*x1
b2 ← a2+b3*x1
b1 ← a1+b2*x1
b0 ← a0+b1*x1


b0
c4 ← b4
c3 ← b3+c4*x2
c2 ← b2+c3*x2
c1 ← b1+c2*x2
d ← (-1)*((c3^2)-4*c4*c2)
x3 ← -c3/2
x4 ← -c3/2
x1
x2
x3,'+',d,'i/2'
x4,'-',d,'i/2'
FinAlgoritmo

iii)Imagen de la grafica



iv)Tabla de iteraciones

Por bairstow

1 | 1.1805555556 | -0.6875 | 0.2222222222 | 1.9722222222 | 4.4722222222 | -0.8567812417 | 2.2914335199 | 0.6432187583 | 3.7914335199 | 133.2021541011 | 60.4371277474
1 | -3.2912167072 | 6.4920868505 | -1.4913402611 | 5.8061761622 | -5.2108927411 | 0.3634272781 | -0.7919680968 | 1.0066460364 | 2.999465423 | 36.1027873705 | 26.403641488
2 | -0.6396637354 | 2.0048257292 | -0.7644857049 | 4.0020171494 | 1.0959025275 | 0.0609521833 | -0.5176448035 | 1.0675982197 | 2.4818206195 | 5.7092810928 | 20.8574624391
3 | -0.0619770478 | 0.2741195727 | -0.6425813383 | 3.0073934501 | 1.5539392306 | 0.0010201433 | -0.0916756713 | 1.068618363 | 2.3901449482 | 0.0954637615 | 3.8355695278
4 | -0.0001866013 | 0.0083308317 | -0.6405410517 | 2.8249123858 | 1.4875806893 | -0.0005383538 | -0.0026655648 | 1.0680800092 | 2.3874793834 | 0.0504038886 | 0.1116476567
5 | 0.0000029939 | 0.0000099753 | -0.6416177594 | 2.8191212116 | 1.479200831 | -0.0000016681 | -0.0000026632 | 1.0680783411 | 2.3874767202 | 0.0001561813 | 0.0001115475

v)Determinar valores iniciales

Para bairtow

r0=1.5
s0=1.5

vi)Respuesta

x1=2.1688708303
x2=-1.1007924893
x3=0.8548497184+1.5446851869i/2
x4=0.8548497184-1.5446851869i/2

Ejercicio3

$$f(x) = \cos(x) - \text{sen}(x) + 0.9$$

i)Algoritmo

Por punto fijo:

```
from sympy import *
def g(x):
      return acos(sin(x)-0.9)

def dx(x):
      return -cos(x)/sqrt(1.8*sin(x)-(pow(sin(x),2))+0.19)

print "Para f(X)=-sen(x)+cos(x)+0.9"
print "G(x)=(cos(sen(x)+0.9))^1"
print "1. Determinar valor inicial"
print "2. Determinar rango"


op=int(input ("Ingrese la opcion"))
errorEspecifico=0.05
errorAproximado=1
gx=0

if op==1:
      r=float(input("Ingrese el valor inicial"))
      if dx(r) >1 :
            print "No hay convergencia"
            pass
elif op==2 :
      val1=float(input("Ingrese el primer valor"))
      val2=float(input("Ingrese el segundo valor"))
      if g(val1)>val1 and g(val2)<val2:
            print "valores: ",val1," y ",val2
            r=float(input("Ingrese el valor entre val1 y val2"))
            if (r>=val1 and r<=val2):
                  r=r
            else:
                  print "No existe ese valor en el rango"
      else:
            print "No hay convergencia"
else:
      print "Ha ingresado valores incorrectos"
      errorAproximado=0
      pass

if (dx(r))<1:
      iterador=1

      entrada=r

      while errorAproximado>errorEspecifico:
            salida=g(entrada)

            print iterador," | ",entrada," | ",salida," | ",errorAproximado

            errorAproximado=Abs((salida-entrada)/salida)*100
            entrada=salida
            iterador=iterador+1
```

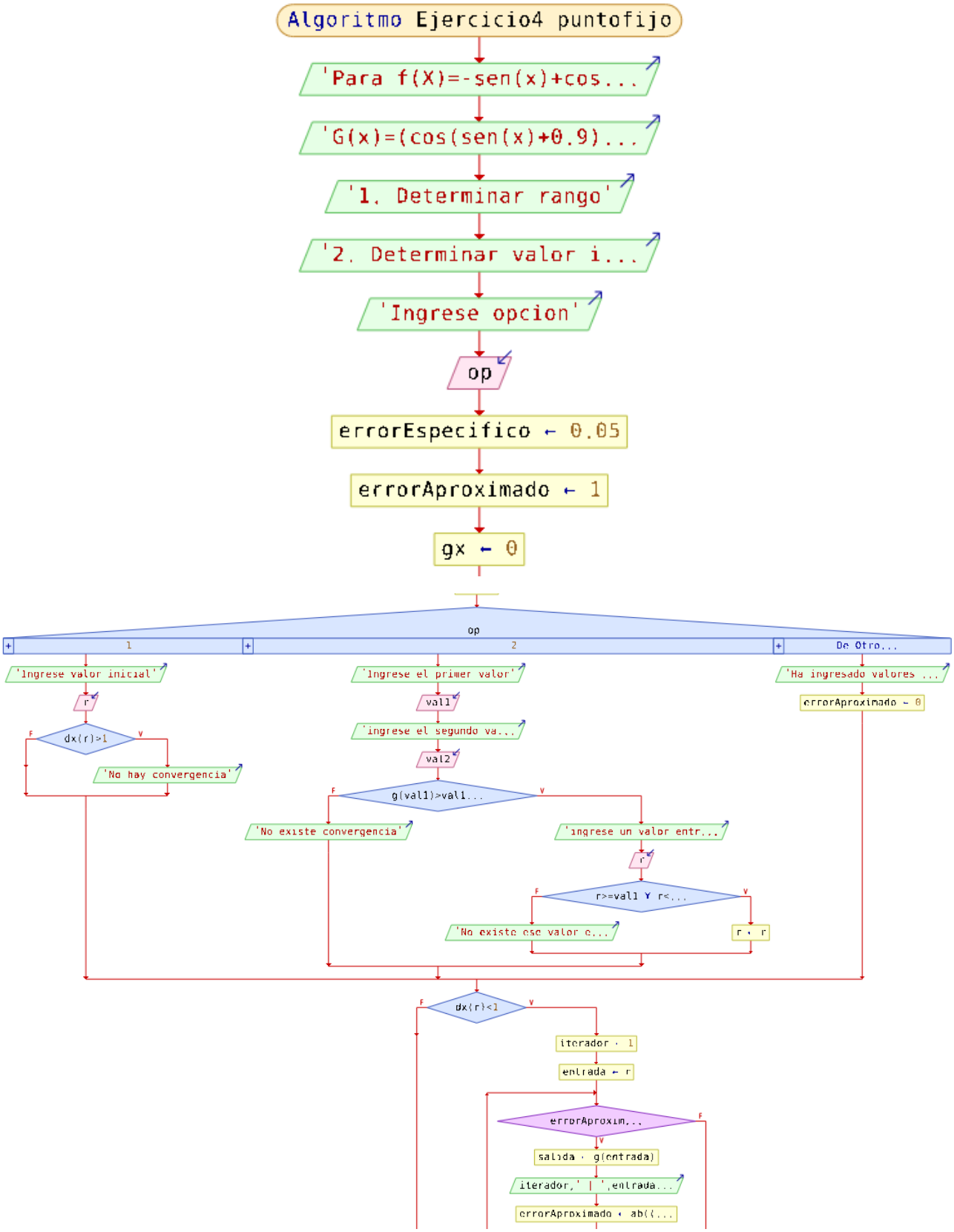
```
      print "La raiz es: ",salida
      print "Con error: ",errorAproximado
```

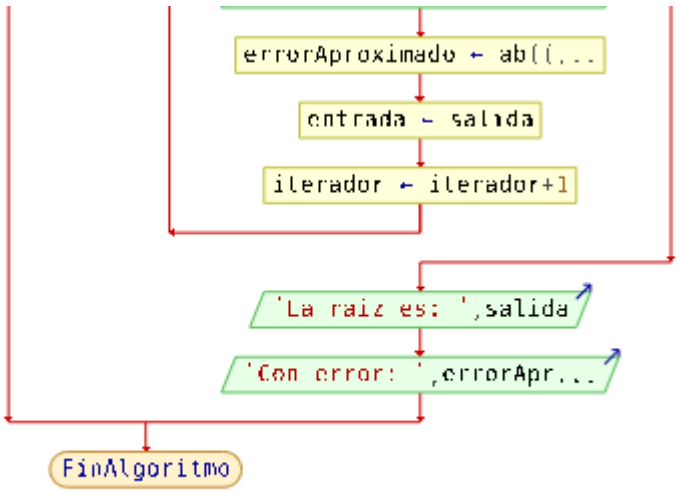
Por el metodo de la secante:

```
from sympy import *

def f(x):
      return cos(x)-sin(x)+0.9

def secante(x1,x2):
      return x2-(f(x2)*(x1-x2)/(f(x1)-f(x2)))

def solucion(x1,x2):
           errorEspecifico=0.05
           errorAproximado=1

           if f(x1)*f(x2)<0:
                print "Existe raiz para f(X)=-sen(x)+cos(x)+0.9"
                itertador=1
                sec=0
                anterior=0
                actual=0
                sec=secante(x1,x2)
                print itertador," | ",x1," | ",x2," | ",f(x1)," | ",f(x2),"
| ",sec," | -----------------"

                x1=x2
                x2=sec

                while errorAproximado>errorEspecifico:
                     anterior=sec
                     sec=secante(x1,x2)
                     actual=sec
                     x1=x2
                     x2=sec
                     errorAproximado=Abs((actual-anterior)/actual)*100
                     print itertador," | ",x1," | ",x2," | ",f(x1)," |
",f(x2)," | ",sec," | ",errorAproximado
                     itertador=itertador+1

                print "La raiz es: ",sec
                print "Con error ",errorAproximado
           else:
                print "No existe raiz en el rango ingresado"


print "Ingrese dos numero"

x1=float(input("valor 1\n"))
x2=float(input("valor 2\n"))

solucion(x1,x2)
```

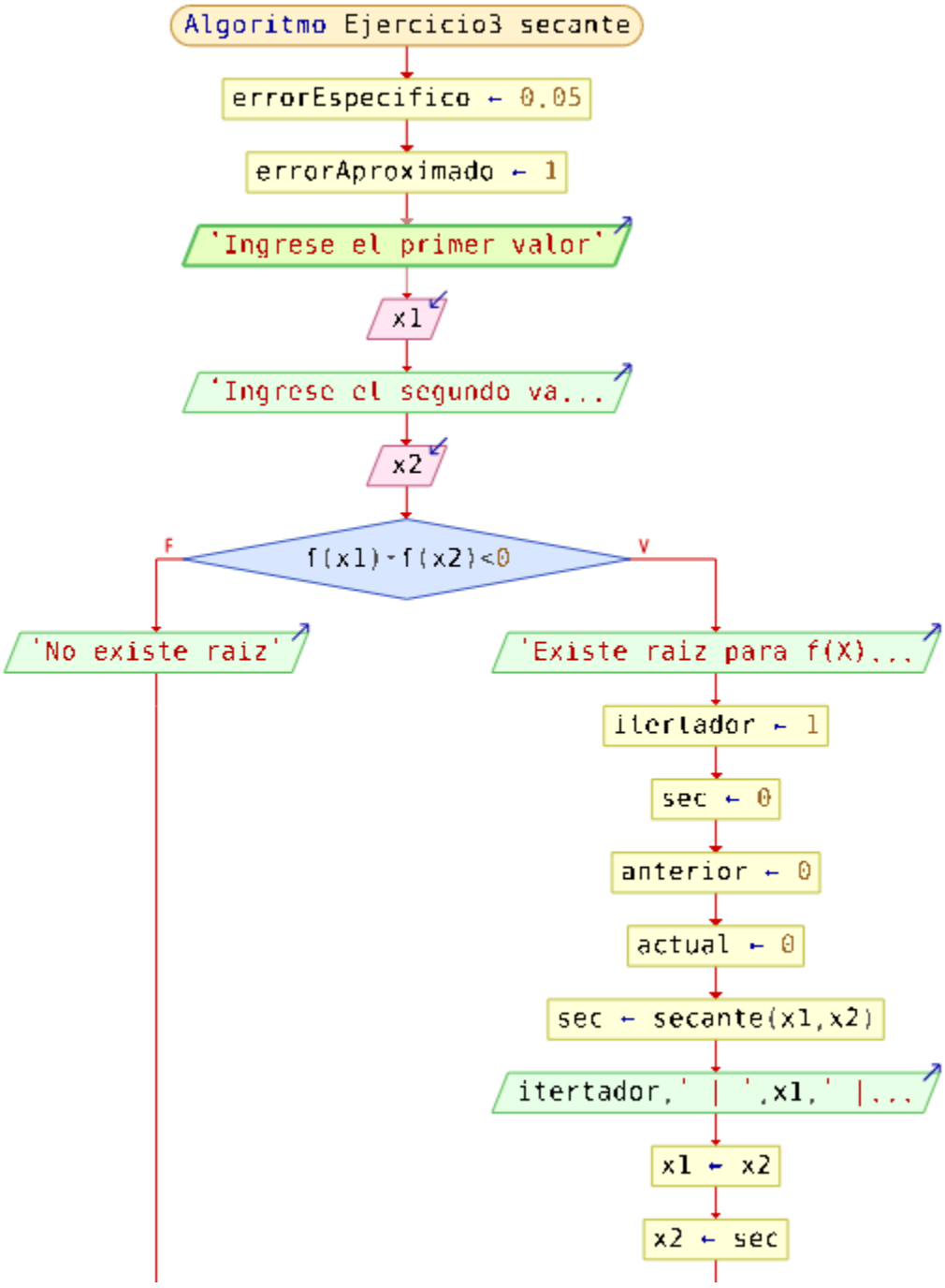
ii)Flujogramas

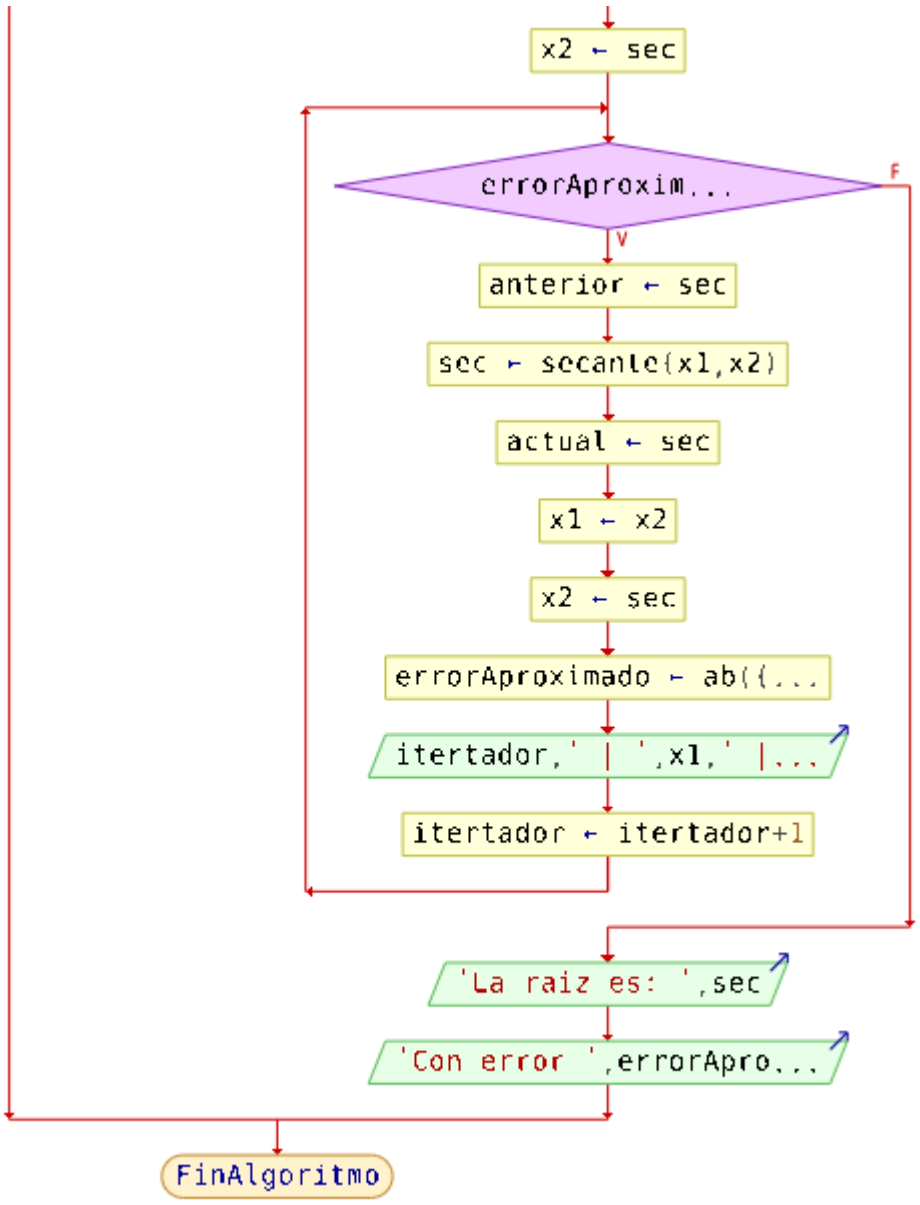
Por punto fijo:



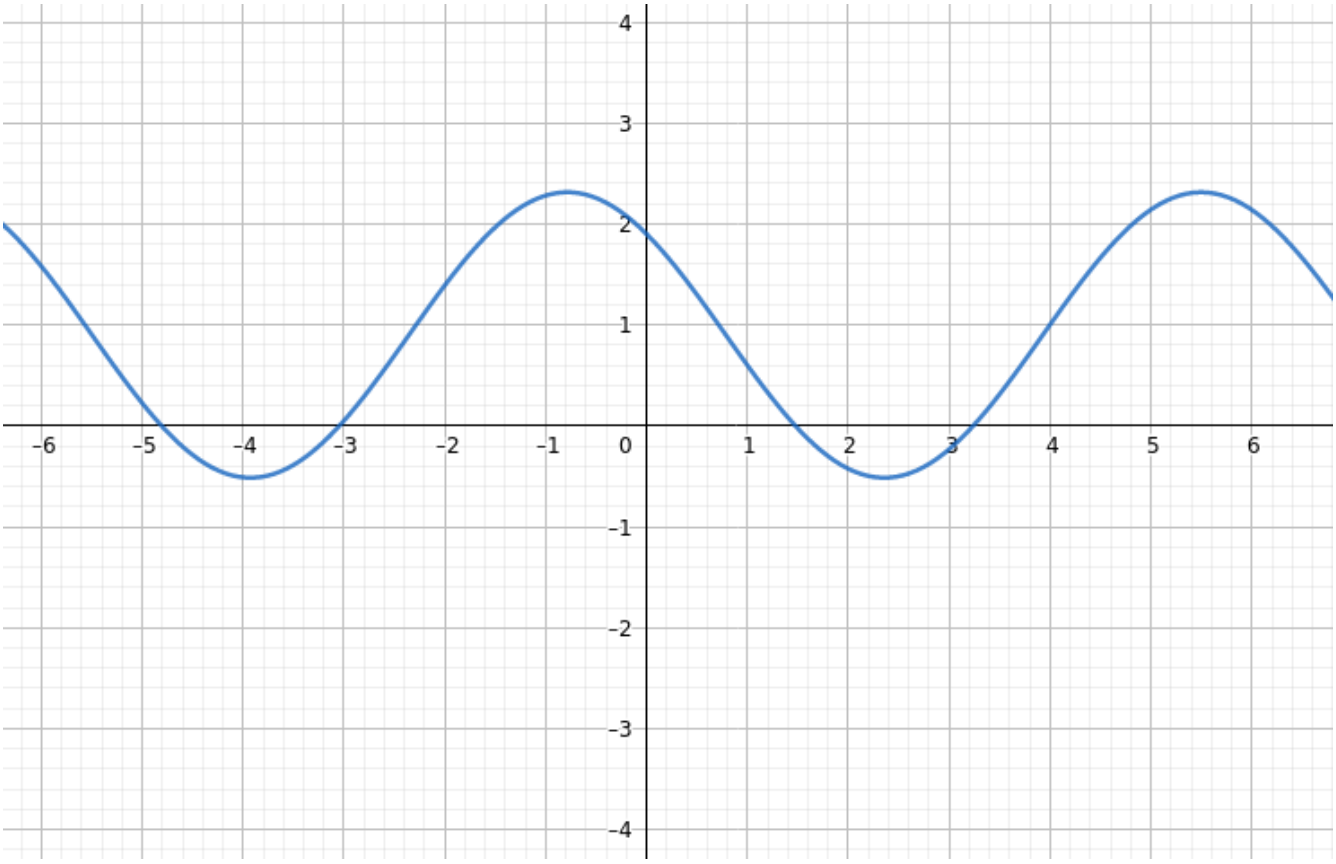

errorAproximado ← ab(...
entrada ← salida
iterador ← iterador+1
'La raiz es: ',salida
'Con error: ',errorApr...
FinAlgoritmo

Por el metodo de la secante:




x2 ← sec
errorAproxim...
F
V
anterior ← sec
sec ← secante(x1,x2)
actual ← sec
x1 ← x2
x2 ← sec
errorAproximado ← ab((...
itertador,' | ',x1,' |...
itertador ← itertador+1
'La raiz es: ',sec
'Con error ',errorApro...
FinAlgoritmo

iii) Imagen de la grafica



iv) Determinar los valores iniciales

Sugerir 1.4 o valores entre 1 y 2 para el metodo de punto fijo
Sugieron valores entre 1 y 2 para el metodo de la secante

v) Tabla de iteraciones

Por punto fijo

1	1.4	1.4852422662	1
2	1.4852422662	1.4743041779	5.7392836273
3	1.4743041779	1.4753030159	0.7419153024
4	1.4753030159	1.4752068433	0.0677039224

Por el metodo de la secante:

1	1	2	0.5988313211	-0.4254442634	1.5846388708	----------
1	1.5846388708	1.4330633247	-0.1137462954	0.0467681451	1.4330633247	10.5770305837
2	1.4330633247	1.4772269968	0.0467681451	-0.0021927302	1.4772269968	2.9896334325
3	1.4772269968	1.4752491111	-0.0021927302	-0.0000369337	1.4752491111	0.1340713045
4	1.4752491111	1.4752152254	-0.0000369337	0.0000000307	1.4752152254	0.0022970014

vi)Respuesta:

Por punto fijo:

La raiz es: 1.4752068433
Con error: 0.0065192624

Por secante:

La raiz es: 1.4752152254
Con error 0.0022970014

Ejercicio4

$$f(x) = -x^3 + 2x^2 + x - 1$$

i)Algoritmo

Por tartaglia

```
from sympy import *
from math import *
def solucion():

     print "Solucion para -x^3+2x^2+x-1=0, pero dividimos toda la ecuacion
por -1"
     print "ecuacion a resolver -x^3+2x^2+x-1=0"

     errorAproximado=1
     errorEspecifico=0.05
     a=-2
     b=-1
     c=1
     k=-a/3
     p=b-pow(a,2)/3



     q=c-(a*b)/3+(2*(pow(a,3)/27))

     D=(4*pow(p,3)+27*pow(q,2))/108

     if D>0:
          A=-q/2+sqrt(pow(q,2)/4+pow(q,3)/27)
          B=-q/2-sqrt(pow(q,2)/4+pow(q,3)/27)
          x1=pow(A,(1/3))+pow(B,(1/3))
          x2=-(pow((A),(1/3))+pow((A),(1/3)))/2+pow(3,(1/3))*(pow((A),
(1/3))-pow((A),(1/3)))/2
          x3=-(pow((A),(1/3))+pow((A),(1/3)))/2-pow(3,(1/3))*(pow((A),
(1/3))-pow((A),(1/3)))/2
     elif D<0:
          cosfi=((sqrt(27))*q)/((2*p)*(sqrt(-p)))
          fi=acos(cosfi)

          x1=(2*sqrt(-p/3))*(cos(fi/3))+k
          x2=(2*sqrt(-p/3))*(cos(fi/3+(2*pi/3)))+k
          x3=(2*sqrt(-p/3))*(cos(fi/3+(4*pi/3)))+k


     else:
          x1=2*pow((-(q/2)),(1/3))+k
          x2=pow((q/2),(1/3))+k
          x3=pow((q/2),(1/3))+k


     print "Las raices son: \n"
     print x1
     print x2
     print x3
solucion()
```

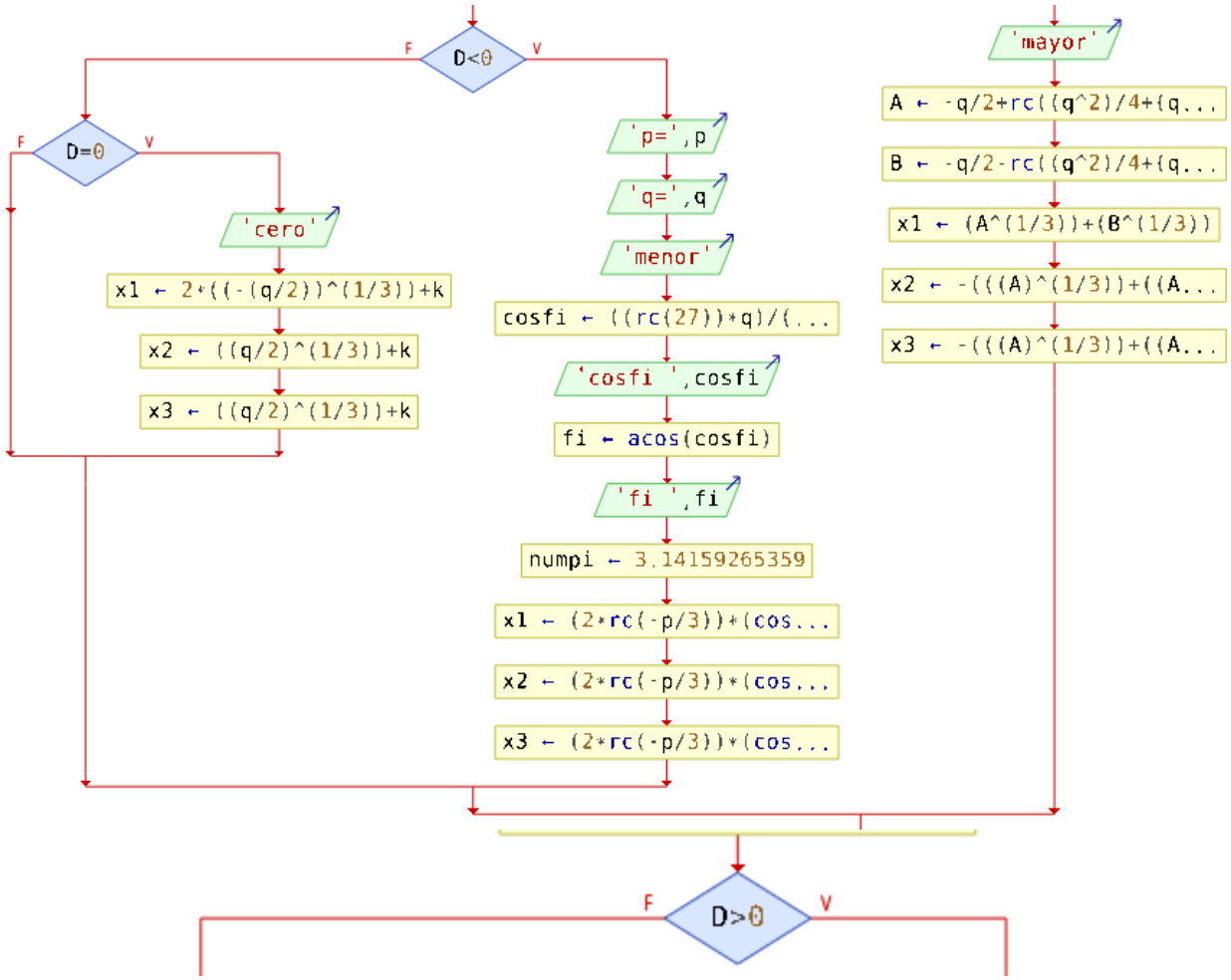
Por horner

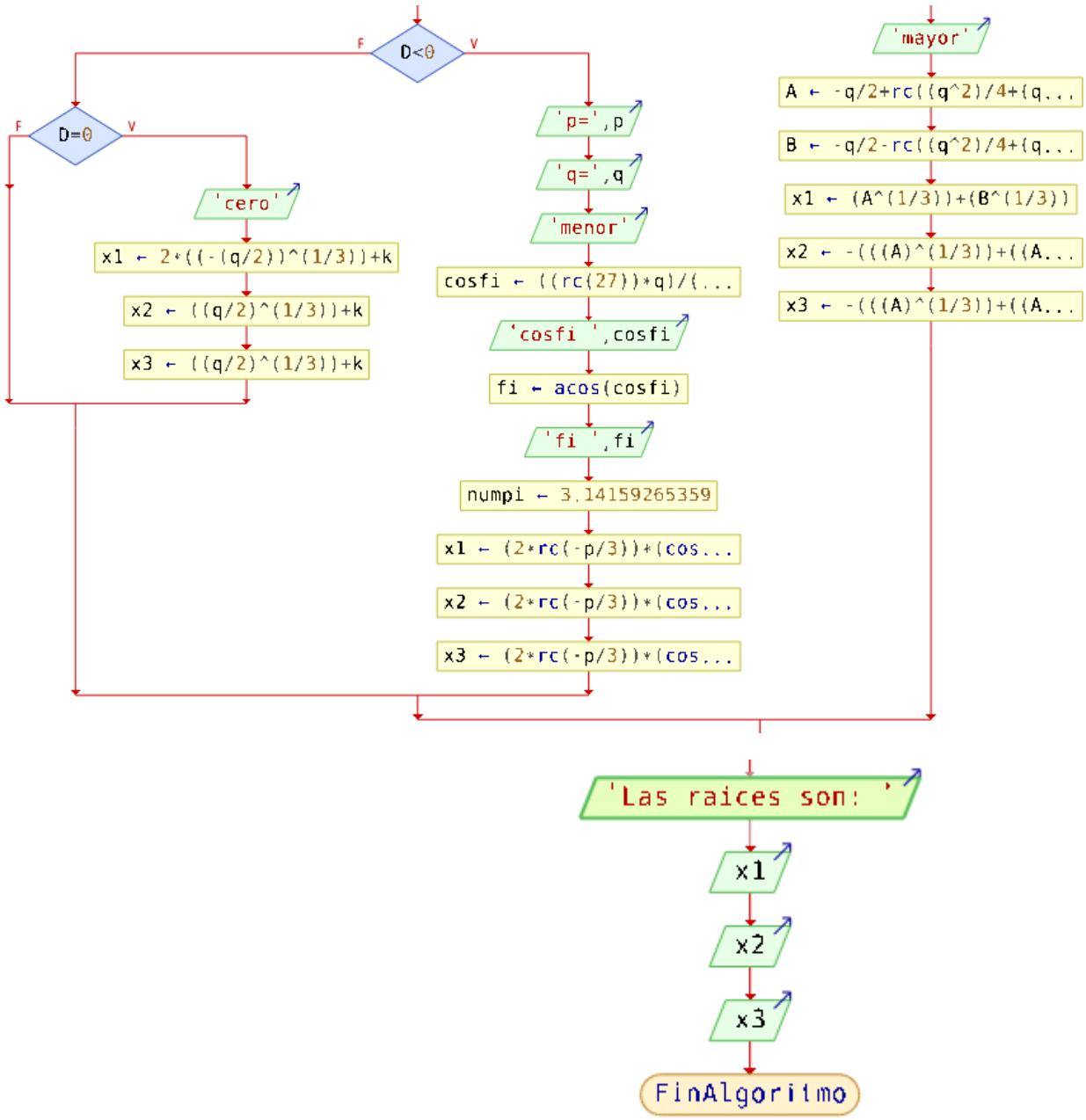
```
from sympy import *

def horner(x0,r0,s0):
      return x0-(r0/s0)

def solucion(x0):


      errorAproximado=1
      errorEspecifico=0.05

      a0=1
      a1=-1
      a2=-2
      a3=1
      r0=0
      r1=0
      r2=0
      r3=0
      s2=0
      s1=0
      s0=0
      iterador=1

      r3=a3
      r2=a2+r3*x0
      r1=a1+r2*x0
      r0=a0+r1*x0

      s2=r3
      s1=r2+s2*x0
      s0=r1+s1*x0
      hor=horner(x0,r0,s0)
      errorAproximado=Abs((x0-hor)/hor)*100

      while errorAproximado>errorEspecifico:
            x0=hor
            r3=a3
            r2=a2+r3*x0
            r1=a1+r2*x0
            r0=a0+r1*x0

            s2=r3
            s1=r2+s2*x0
            s0=r1+s1*x0
            hor=horner(x0,r0,s0)
            errorAproximado=Abs((x0-hor)/hor)*100

            print iterador," | ",hor," | ",errorAproximado
            iterador=iterador+1

      print "La raiz es: ",hor
      print "Con error aproximado: ",errorAproximado

print "Solucion para -x^3+2x^2+x-1"

x0=float(input("Ingrese el valor inicial"))
```

```
solucion(x0)
```

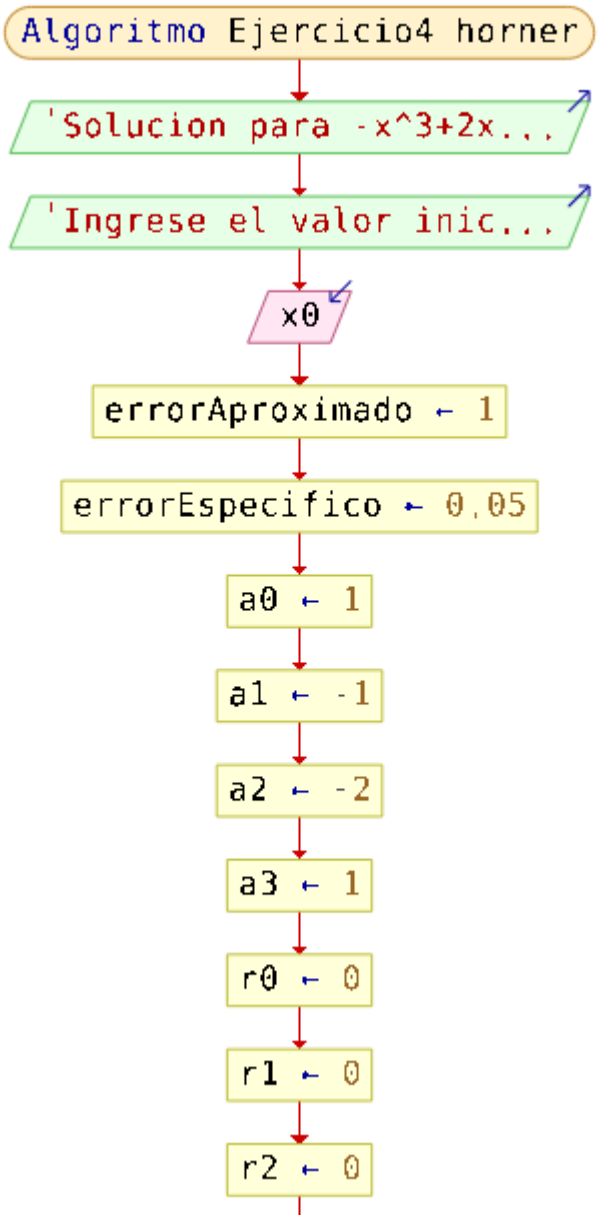
ii)Flujograma

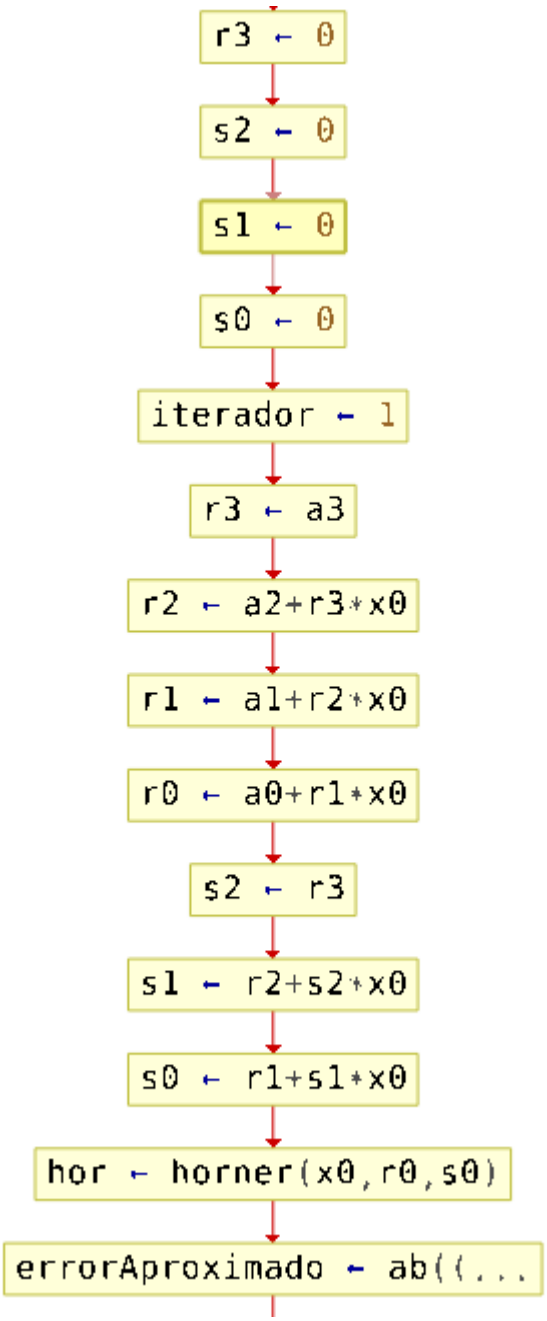
Por Tartaglia:

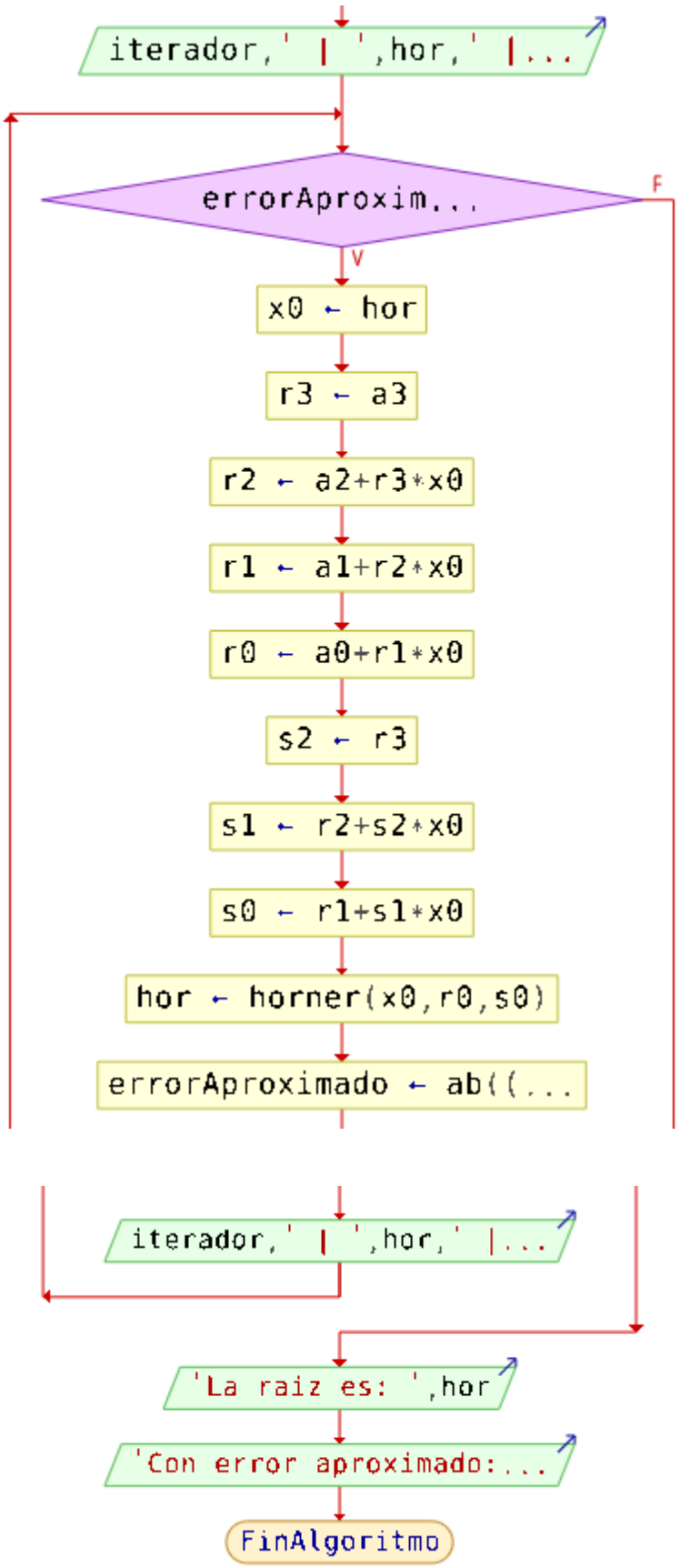



D<0
F
V
D=0
F
V
'cero'
x1 ← 2*((-(q/2))^(1/3))+k
x2 ← ((q/2)^(1/3))+k
x3 ← ((q/2)^(1/3))+k
'p=',p
'q=',q
'menor'
cosfi ← ((rc(27))*q)/(...
'cosfi ',cosfi
fi ← acos(cosfi)
'fi ',fi
numpi ← 3.14159265359
x1 ← (2*rc(-p/3))*(cos...
x2 ← (2*rc(-p/3))*(cos...
x3 ← (2*rc(-p/3))*(cos...
'mayor'
A ← -q/2+rc((q^2)/4+(q...
B ← -q/2-rc((q^2)/4+(q...
x1 ← (A^(1/3))+(B^(1/3))
x2 ← -(((A)^(1/3))+((A...
x3 ← -(((A)^(1/3))+((A...
'Las raices son: '
x1
x2
x3
FinAlgoritmo

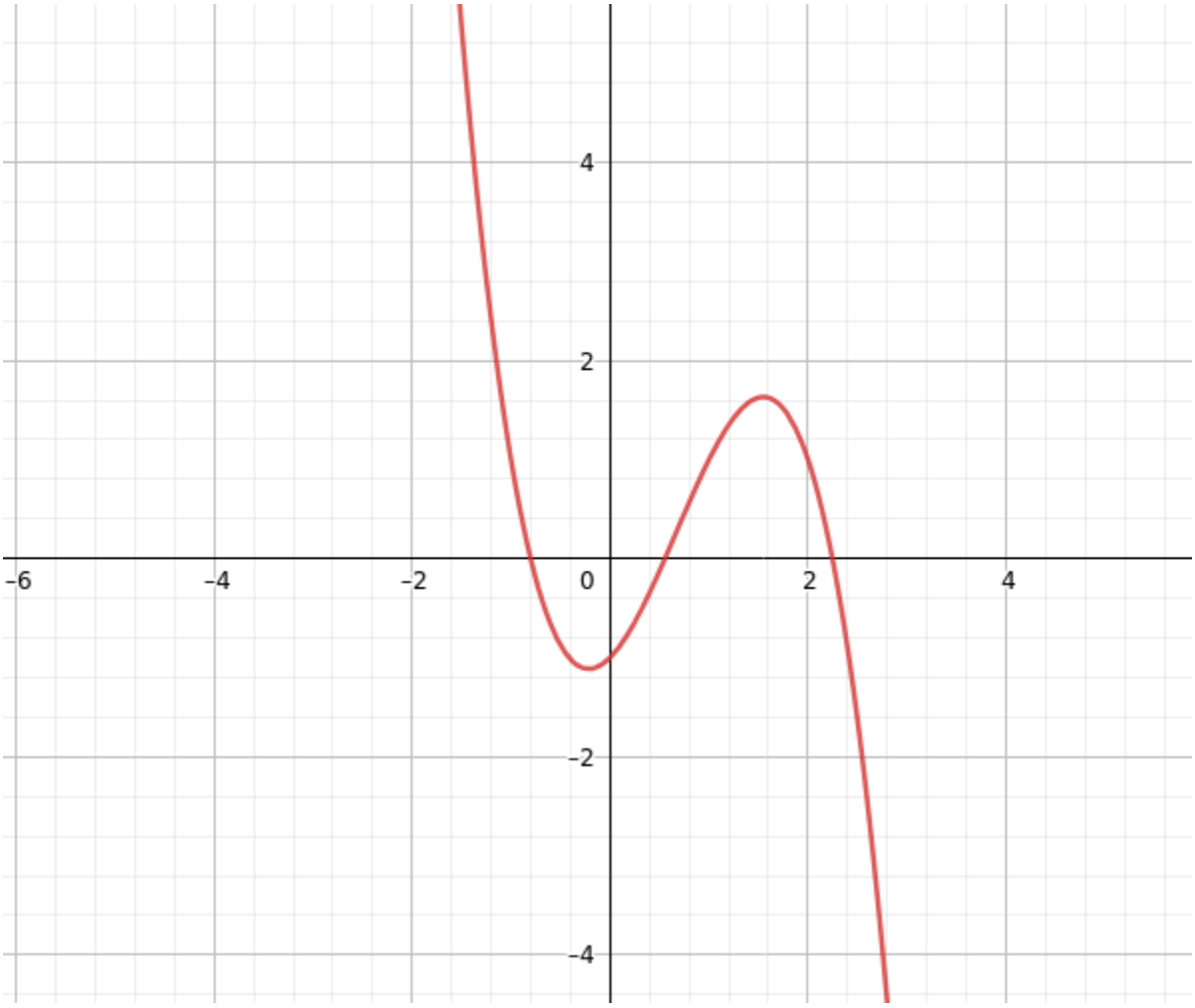
Por Horner


Algoritmo Ejercicio4 horner
'Solucion para -x^3+2x...
'Ingrese el valor inic...
x0
errorAproximado ← 1
errorEspecifico ← 0,05
a0 ← 1
a1 ← -1
a2 ← -2
a3 ← 1
r0 ← 0
r1 ← 0
r2 ← 0


r3 ← 0
s2 ← 0
s1 ← 0
s0 ← 0
iterador ← 1
r3 ← a3
r2 ← a2+r3*x0
r1 ← a1+r2*x0
r0 ← a0+r1*x0
s2 ← r3
s1 ← r2+s2*x0
s0 ← r1+s1*x0
hor ← horner(x0,r0,s0)
errorAproximado ← ab((...


iterador,' | ',hor,' |...
errorAproxim...
F
V
x0 ← hor
r3 ← a3
r2 ← a2+r3*x0
r1 ← a1+r2*x0
r0 ← a0+r1*x0
s2 ← r3
s1 ← r2+s2*x0
s0 ← r1+s1*x0
hor ← horner(x0,r0,s0)
errorAproximado ← ab(...
iterador,' | ',hor,' |...
'La raiz es: ',hor
'Con error aproximado:...
FinAlgoritmo

iii)Imagen de la grafica



iv)Determinar valores iniciales

Se sugiere valor inicial 2 para el metodo de horner

v)Tabla de iteraciones

1	2.3333333333	14.2857142857
1	2.2530864198	3.5616438356
2	2.2470135821	0.270262613
3	2.2469796048	0.0015121345

vi)Respuesta

La raiz es: 2.2469796048
Con error aproximado: 0.001512134

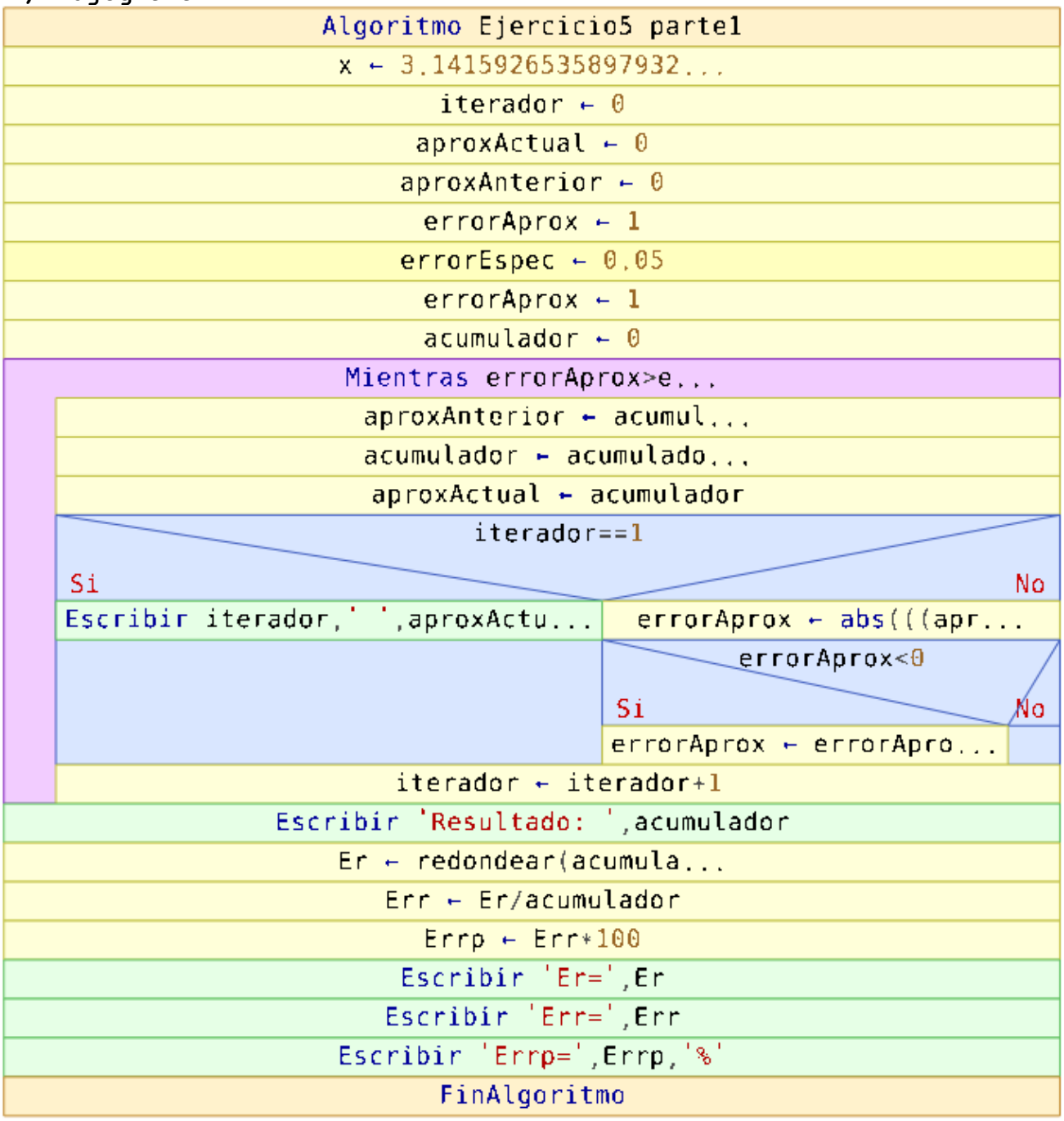
Ejercicio5

$$\sinh(x)$$

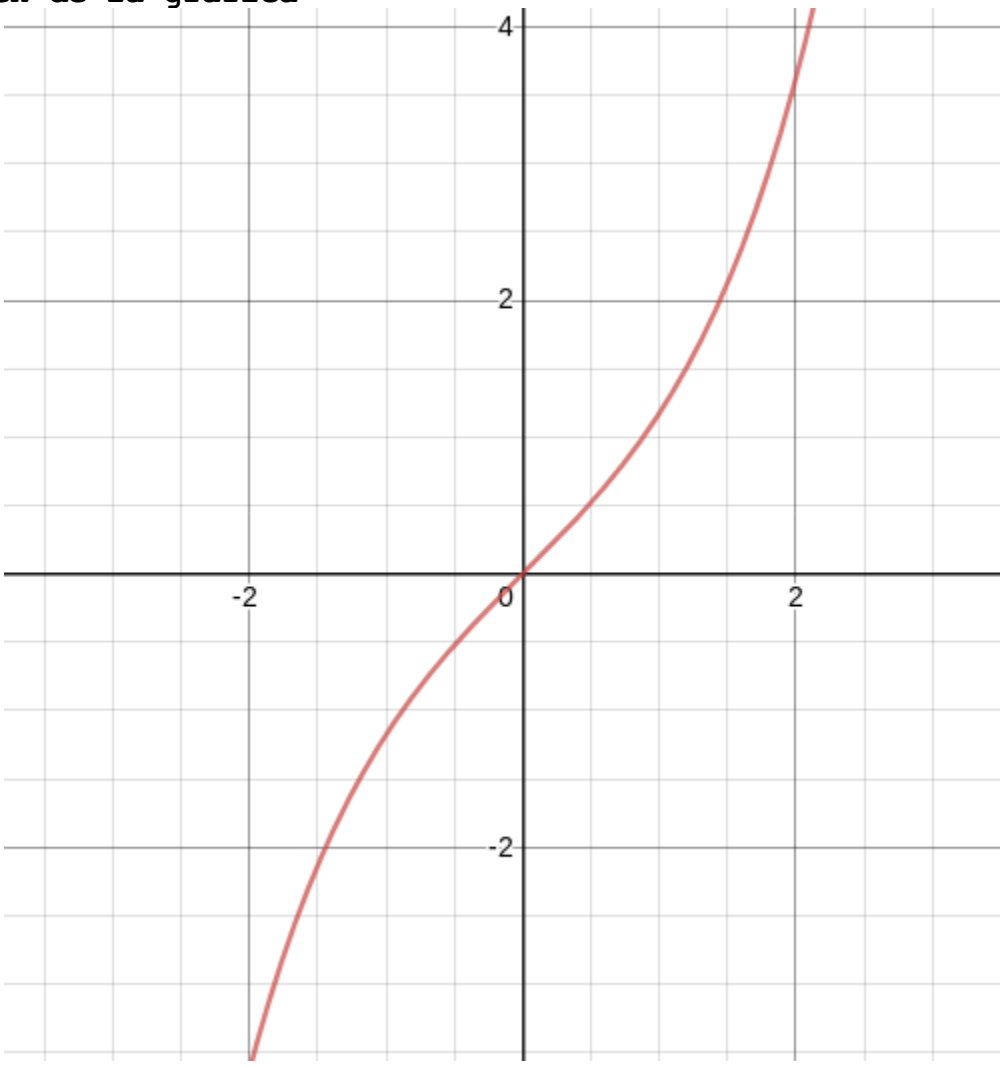
i)Algoritmo

```
from sympy import *
from math import *
from decimal import Decimal

def truncate(number):

    return trunc(number*100000)/100000;

x=pi/9
iterador=0
aproxActual=0
aproxAnterior=0
errorAprox=1
errorEspec=0.05
errorAprox=1
acumulador=0.0
actual=0
while (errorAprox>errorEspec) :
     aproxAnterior=acumulador

acumulador=acumulador+(pow(x,(2*iterador+1)))/(factorial(2*iterador+1))
     aproxActual=acumulador

     if iterador==1:
          print(iterador,' ',aproxActual,' ','----')
     else:
          errorAprox=Abs(((aproxActual-aproxAnterior)/aproxActual)*100);

     iterador=iterador+1

print("Resultado: ",acumulador)
print("Valor al redondear hasta 4 decimales :",round(acumulador,4))
Er=Abs(acumulador-round(acumulador,4))
Err=Er/acumulador
Errp=Err*100
print "Er =",Er
print "Err =",Err
print "Errp =",Errp

print("Valor al truncar hasta 5 decimales :",round(acumulador))
Er=acumulador-truncate(acumulador)
Err=Er/acumulador
Errp=Err*100
print "Er =",Er
print "Err =",Err
print "Errp =",Errp
```

ii)Flujograma



iii)Imagen de la grafica



v)Respuesta

```
('Resultado: ', 0.3561978068964175)
('Valor al redondear hasta 4 decimales :', 0.3562)
Er = 2.19310358251512e-6
Err = 6.15698227236103e-6
Errp = 0.000615698227236103
('Valor al truncar hasta 5 decimales :', 0.0)
Er = 0.356197806896
Err = 1.0
Errp = 100.0
```