McMullen's_example_by_McMullen_algo

March 14, 2022

[386]:
```
__author__="William Huanshan Chuang"
__version__="1.0.1"
__email__="wchuang2@mail.sfsu.edu"

#define elementary functions
import math
import statistics
def Cartesian_complex_mul(numb1,numb2):
    #numb1 a+bi, [a,b], numb2 c+di, [c,d]
    x=numb1[0]*numb2[0]-numb1[1]*numb2[1]
    y=numb1[0]*numb2[1]+numb1[1]*numb2[0]
    return [x,y]

def Cartesian_complex_scalar_mul(alpha,numb1):
    #numb1 a+bi, [a,b], numb2 c+di, [c,d]
    x=numb1[0]*alpha
    y=numb1[1]*alpha
    return [x,y]

def Cartesian_complex_add(numb1,numb2):
    #numb1 a+bi, [a,b], numb2 c+di, [c,d]
    x=numb1[0]+numb2[0]
    y=numb1[1]+numb2[1]
    return [x,y]

def Cartesian_complex_divide(numb1,numb2):
    #numb1 u+vi, [u,v], numb2 x+yi, [x,y]
    d=numb2[0]*numb2[0]+numb2[1]*numb2[1]
    nx=numb1[0]*numb2[0]+numb1[1]*numb2[1]
    ny=numb1[1]*numb2[0]-numb1[0]*numb2[1]
    X=float(nx/d)
    Y=float(ny/d)
    return[X,Y]

def Cartesian_complex_modulus(numb):
    return math.sqrt(numb[0]*numb[0]+numb[1]*numb[1])
```

```
def Cartesian_complex_complex_conjugate(numb):
    return [numb[0],-numb[1]]

def Cartesian_complex_complex_to_polar(numb):
    r=Cartesian_complex_modulus(numb)
    if numb[0]>0:
        t=math.atan(float(numb[1]/numb[0]))
    elif numb[0]<0:
        t=math.atan(float(numb[1]/numb[0]))+math.pi
    else:
        if numb[1]>0:
            t=float(math.pi/2)
        elif numb[1]<0:
            t=float(-math.pi/2)
        else:
            t="null"
    return [r,t]

def Polar_complex_complex_to_Cartesian(numb):
    return [numb[0]*math.cos(numb[1]),numb[0]*math.sin(numb[1])]

def Polar_complex_conjugate(numb):
    return [numb[0],-numb[1]]

def Polar_complex_mul(numb1,numb2):
    #numb1 r_1e^(it_1), [r_1,t_1], numb2 r_2e^(it_2), [r_2,t_2]
    r=numb1[0]*numb2[0]
    t=numb1[1]+numb2[1]
    return [r,t]

def Polar_complex_divide(numb1,numb2):
    #numb1 r_1e^(it_1), [r_1,t_1], numb2 r_2e^(it_2), [r_2,t_2]
    r=float(numb1[0]/numb2[0])
    t=numb1[1]-numb2[1]
    return [r,t]

def Polar_complex_add(numb1,numb2):
    N1=Polar_complex_complex_to_Cartesian(numb1)
    N2=Polar_complex_complex_to_Cartesian(numb2)
    tot=Cartesian_complex_add(N1,N2)
    return Cartesian_complex_complex_to_polar(tot)

def real_matrix_addition(m1,m2):
    #m1=[[M11,M12],[M21,M22]]
    a=m1[0][0]+m2[0][0]
    b=m1[0][1]+m2[0][1]
    c=m1[1][0]+m2[1][0]
```

```
    d=m1[1][1]+m2[1][1]
    l1=[a,b]
    l2=[c,d]
    l=[]
    l.append(l1)
    l.append(l2)
    return l

def Cartesian_complex_matrix_addition(m1,m2):
    #m1=[[M11,M12],[M21,M22]]
    #M11=[a,b]
    a1=m1[0][0][0]+m2[0][0][0]
    a2=m1[0][0][1]+m2[0][0][1]
    b1=m1[0][1][0]+m2[0][1][0]
    b2=m1[0][1][1]+m2[0][1][1]
    c1=m1[1][0][0]+m2[1][0][0]
    c2=m1[1][0][1]+m2[1][0][1]
    d1=m1[1][1][0]+m2[1][1][0]
    d2=m1[1][1][1]+m2[1][1][1]
    a=[a1,a2]
    b=[b1,b2]
    c=[c1,c2]
    d=[d1,d2]
    l1=[a,b]
    l2=[c,d]
    l=[]
    l.append(l1)
    l.append(l2)
    return l

def real_matrix_multiplication(m1,m2):
    #m1=[[M11,M12],[M21,M22]]
    a=m1[0][0]*m2[0][0]+m1[0][1]*m2[1][0]
    b=m1[0][0]*m2[0][1]+m1[0][1]*m2[1][1]
    c=m1[1][0]*m2[0][0]+m1[1][1]*m2[1][0]
    d=m1[1][0]*m2[0][1]+m1[1][1]*m2[1][1]
    l1=[a,b]
    l2=[c,d]
    l=[]
    l.append(l1)
    l.append(l2)
    return l

def Cartesian_complex_matrix_multiplication(m1,m2):
    #m1=[[M11,M12],[M21,M22]]
    #M11=[a,b]
```

```
↪a=Cartesian_complex_add(Cartesian_complex_mul(m1[0][0],m2[0][0]),Cartesian_complex_mul(m1[0
↪b=Cartesian_complex_add(Cartesian_complex_mul(m1[0][0],m2[0][1]),Cartesian_complex_mul(m1[0
↪c=Cartesian_complex_add(Cartesian_complex_mul(m1[1][0],m2[0][0]),Cartesian_complex_mul(m1[1
↪d=Cartesian_complex_add(Cartesian_complex_mul(m1[1][0],m2[0][1]),Cartesian_complex_mul(m1[1
    l1=[a,b]
    l2=[c,d]
    l=[]
    l.append(l1)
    l.append(l2)
    return l

def real_matrix_inverse(m1):
    #m1=[[M11,M12],[M21,M22]]
    det=m1[0][0]*m1[1][1]-m1[0][1]*m1[1][0]
    a=float(m1[1][1]/det)
    b=float(-m1[0][1]/det)
    c=float(-m1[1][0]/det)
    d=float(m1[0][0]/det)
    l1=[a,b]
    l2=[c,d]
    l=[]
    l.append(l1)
    l.append(l2)
    return l

def Cartesian_complex_matrix_inverse(m1):
    #m1=[[M11,M12],[M21,M22]]
    #M11=[a,b]
↪det=Cartesian_complex_add(Cartesian_complex_mul(m1[0][0],m1[1][1]),Cartesian_complex_scalar
    inverse_det=Cartesian_complex_divide([1,0],det)
    a=Cartesian_complex_mul(m1[1][1],inverse_det)
↪b=Cartesian_complex_mul(Cartesian_complex_scalar_mul(-1,m1[0][1]),inverse_det)
↪c=Cartesian_complex_mul(Cartesian_complex_scalar_mul(-1,m1[1][0]),inverse_det)
    d=Cartesian_complex_mul(m1[0][0],inverse_det)
    l1=[a,b]
    l2=[c,d]
    l=[]
    l.append(l1)
    l.append(l2)
```

```
    return l


def Cartesian_radial_hyperbolic_distance(z):
    r=float(Cartesian_complex_modulus(z))
    return math.log(float((1+r)/(1-r)))

def operator_T(Lambda):
    D=2
    a1=(-Lambda-float(1/Lambda))*float(-0.5)
    a2=0
    b1=0
    b2=(-Lambda+float(1/Lambda))*float(-0.5)
    c1=0
    c2=(Lambda-float(1/Lambda))*float(-0.5)
    d1=(-Lambda-float(1/Lambda))*float(-0.5)
    d2=0
    l1=[[a1,a2],[b1,b2]]
    l2=[[c1,c2],[d1,d2]]
    l=[]
    l.append(l1)
    l.append(l2)
    return l

def operator_R(theta):
    a=Polar_complex_complex_to_Cartesian([1,float(0.5*theta)])
    b=[0,0]
    c=[0,0]
    d=Polar_complex_complex_to_Cartesian([1,float(-0.5*theta)])
    l1=[a,b]
    l2=[c,d]
    l=[]
    l.append(l1)
    l.append(l2)
    return l

def classification_point(Lambda):
    return [float((2*Lambda**2)/(Lambda**4+1)),float((Lambda**4-1)/
 ↪(Lambda**4+1))]

def check_T_generate_a_Schottky(Lambda,m):
    t=float(-math.pi/2)+float(math.pi/(2*m))
    K=Polar_complex_complex_to_Cartesian([1,t])
    B=classification_point(Lambda)
    T=operator_T(Lambda)
    T0=Cartesian_complex_divide(T[0][1],T[1][1])
```

```
    ␣
  ↪discriminant=float(Cartesian_complex_modulus(Cartesian_complex_add(K,Cartesian_complex_scal
    print(discriminant)
    if discriminant>0:
        return True
    else:
        return False

def Tz(T,z):
    a=T[0][0]
    b=T[0][1]
    c=T[1][0]
    d=T[1][1]
    return ␣
  ↪Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_mul(a,z),b),Cartesian_comp




#Generate the orbit Gamma(0)
def Gamma0(Lambda,m,N):
    T=operator_T(Lambda)
    theta=float(math.pi/m)
    R=operator_R(theta)
    L=[Tz(T,[0,0])]
    tmp1=[]
    tmp2=[]
    nodes_in_DT=[[0,0]]
    j=1
    while j<=N:
        #N=1
        if j==1:
            z=L[0]
            tmp1=[]
            tmp2=[]
            for i in range(2*m-1):
                z=Tz(R,z)
                tmp1.append(z)
                if i!=m-1:
                    tmp2.append(z)
                else:
                    tmp2.append(L[0])
            L=[]
            for k in tmp2:
                L.append(k)
            nodes_in_DT=[Tz(T,[0,0])]
```

```
        #N>1
        else:
            nodes_in_DT=[]
            tmp1=[]
            tmp2=[]
            tmp3=[]
            for k in L:
                z=Tz(T,k)
                nodes_in_DT.append(z)
                tmp1.append(z)
                tmp2.append(z)
            L=[]
            for i in range(2*m-1):
                if i!=m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                        L.append(k)
                    tmp3=[]
                elif i==m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                    tmp3=[]
                    for k in tmp2:
                        L.append(k)

        j+=1
    return nodes_in_DT
# measuring hyperbolic distance
def Hyperbolic_Distance_Gamma0(Lambda,m,N):
    T=operator_T(Lambda)
    theta=float(math.pi/m)
    R=operator_R(theta)
    L=[Tz(T,[0,0])]
    tmp1=[]
    tmp2=[]
    nodes_in_DT=[[0,0]]
    j=1
    while j<=N:
        #N=1
        if j==1:
            z=L[0]
```

```
        tmp1=[]
        tmp2=[]
        for i in range(2*m-1):
            z=Tz(R,z)
            tmp1.append(z)
            if i!=m-1:
                tmp2.append(z)
            else:
                tmp2.append(L[0])
        L=[]
        for k in tmp2:
            L.append(k)
        nodes_in_DT=[Tz(T,[0,0])]

    #N>1
    else:
        nodes_in_DT=[]
        tmp1=[]
        tmp2=[]
        tmp3=[]
        for k in L:
            z=Tz(T,k)
            nodes_in_DT.append(z)
            tmp1.append(z)
            tmp2.append(z)
        L=[]
        for i in range(2*m-1):
            if i!=m-1:
                for k in tmp1:
                    tmp3.append(Tz(R,k))
                tmp1=[]
                for k in tmp3:
                    tmp1.append(k)
                    L.append(k)
                tmp3=[]
            elif i==m-1:
                for k in tmp1:
                    tmp3.append(Tz(R,k))
                tmp1=[]
                for k in tmp3:
                    tmp1.append(k)
                tmp3=[]
                for k in tmp2:
                    L.append(k)

    j+=1
Hyperbolic_distance=[]
```

```
    for k in nodes_in_DT:
        Hyperbolic_distance.append(Cartesian_radial_hyperbolic_distance(k))
    return Hyperbolic_distance

# measuring Exp(-hyperbolic distance)
def Exp_negative_Hyperbolic_Distance_Gamma0(Lambda,m,N):
    T=operator_T(Lambda)
    theta=float(math.pi/m)
    R=operator_R(theta)
    L=[Tz(T,[0,0])]
    tmp1=[]
    tmp2=[]
    nodes_in_DT=[[0,0]]
    j=1
    while j<=N:
        #N=1
        if j==1:
            z=L[0]
            tmp1=[]
            tmp2=[]
            for i in range(2*m-1):
                z=Tz(R,z)
                tmp1.append(z)
                if i!=m-1:
                    tmp2.append(z)
                else:
                    tmp2.append(L[0])
            L=[]
            for k in tmp2:
                L.append(k)
            nodes_in_DT=[Tz(T,[0,0])]

        #N>1
        else:
            nodes_in_DT=[]
            tmp1=[]
            tmp2=[]
            tmp3=[]
            for k in L:
                z=Tz(T,k)
                nodes_in_DT.append(z)
                tmp1.append(z)
                tmp2.append(z)
            L=[]
            for i in range(2*m-1):
                if i!=m-1:
                    for k in tmp1:
```

```
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                        L.append(k)
                    tmp3=[]
                elif i==m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                    tmp3=[]
                    for k in tmp2:
                        L.append(k)

        j+=1
    Hyperbolic_distance=[]
    for k in nodes_in_DT:
        Hyperbolic_distance.append(math.exp(-Cartesian_radial_hyperbolic_distance(k)))
    return Hyperbolic_distance

# measuring Exp(-hyperbolic distance)
def Exp_negative_Hyperbolic_Distance_Gamma0_with_t(Lambda,m,N,t):
    T=operator_T(Lambda)
    theta=float(math.pi/m)
    R=operator_R(theta)
    L=[Tz(T,[0,0])]
    tmp1=[]
    tmp2=[]
    nodes_in_DT=[[0,0]]
    j=1
    while j<=N:
        #N=1
        if j==1:
            z=L[0]
            tmp1=[]
            tmp2=[]
            for i in range(2*m-1):
                z=Tz(R,z)
                tmp1.append(z)
                if i!=m-1:
                    tmp2.append(z)
                else:
                    tmp2.append(L[0])
            L=[]
```

```
            for k in tmp2:
                L.append(k)
            nodes_in_DT=[Tz(T,[0,0])]

        #N>1
        else:
            nodes_in_DT=[]
            tmp1=[]
            tmp2=[]
            tmp3=[]
            for k in L:
                z=Tz(T,k)
                nodes_in_DT.append(z)
                tmp1.append(z)
                tmp2.append(z)
            L=[]
            for i in range(2*m-1):
                if i!=m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                        L.append(k)
                    tmp3=[]
                elif i==m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                    tmp3=[]
                    for k in tmp2:
                        L.append(k)

        j+=1
    Hyperbolic_distance=[]
    for k in nodes_in_DT:
        Hyperbolic_distance.append(math.
↪exp(-t*Cartesian_radial_hyperbolic_distance(k)))
    return Hyperbolic_distance

# measuring Exp(-t*(hyperbolic distance))
def Improved_Exp_negative_Hyperbolic_Distance_Gamma0_with_t(Lambda,m,N,L,t):

    T=operator_T(Lambda)
    theta=float(math.pi/m)
```

```
    R=operator_R(theta)
    if len(L)==0:
        L=[Tz(T,[0,0])]
        j=1
    else:
        j=N
    tmp1=[]
    tmp2=[]
    nodes_in_DT=[[0,0]]

    while j<=N:
        #N=1
        if j==1:
            z=L[0]
            tmp1=[]
            tmp2=[]
            for i in range(2*m-1):
                z=Tz(R,z)
                tmp1.append(z)
                if i!=m-1:
                    tmp2.append(z)
                else:
                    tmp2.append(L[0])
            L=[]
            for k in tmp2:
                L.append(k)
            nodes_in_DT=[Tz(T,[0,0])]

        #N>1
        else:
            nodes_in_DT=[]
            tmp1=[]
            tmp2=[]
            tmp3=[]
            for k in L:
                z=Tz(T,k)
                nodes_in_DT.append(z)
                tmp1.append(z)
                tmp2.append(z)
            L=[]
            for i in range(2*m-1):
                if i!=m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
```

```
                        L.append(k)
                    tmp3=[]
                elif i==m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                    tmp3=[]
                    for k in tmp2:
                        L.append(k)

        j+=1
    Hyperbolic_distance=[]
    for k in nodes_in_DT:
        Hyperbolic_distance.append(math.
↪exp(-t*Cartesian_radial_hyperbolic_distance(k)))
    return [Hyperbolic_distance,L]



# measuring Exp(-hyperbolic distance)
def Improved_Exp_negative_Hyperbolic_Distance_Gamma0(Lambda,m,N,L):

    T=operator_T(Lambda)
    theta=float(math.pi/m)
    R=operator_R(theta)
    if len(L)==0:
        L=[Tz(T,[0,0])]
        j=1
    else:
        j=N
    tmp1=[]
    tmp2=[]
    nodes_in_DT=[[0,0]]

    while j<=N:
        #N=1
        if j==1:
            z=L[0]
            tmp1=[]
            tmp2=[]
            for i in range(2*m-1):
                z=Tz(R,z)
                tmp1.append(z)
                if i!=m-1:
                    tmp2.append(z)
```

```
                else:
                    tmp2.append(L[0])
            L=[]
            for k in tmp2:
                L.append(k)
            nodes_in_DT=[Tz(T,[0,0])]

        #N>1
        else:
            nodes_in_DT=[]
            tmp1=[]
            tmp2=[]
            tmp3=[]
            for k in L:
                z=Tz(T,k)
                nodes_in_DT.append(z)
                tmp1.append(z)
                tmp2.append(z)
            L=[]
            for i in range(2*m-1):
                if i!=m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                        L.append(k)
                    tmp3=[]
                elif i==m-1:
                    for k in tmp1:
                        tmp3.append(Tz(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                    tmp3=[]
                    for k in tmp2:
                        L.append(k)

        j+=1
    Hyperbolic_distance=[]
    for k in nodes_in_DT:
        Hyperbolic_distance.append(math.
↪exp(-Cartesian_radial_hyperbolic_distance(k)))
    return [Hyperbolic_distance,L]
```

```
def examples_of_10000(initial, increment):
    counter=initial
    useful_example=0
    while counter < initial+10000*increment:
        print("***********"+str(counter)+"***********")
        out=[]
        if check_T_generate_a_Schottky(Lambda=counter,m=2):
            try:
                rho=[]
                for i in range(15):
                    sum1=0
                    sum2=0
                    ␣
 ↪test=Exp_negative_Hyperbolic_Distance_Gamma0(Lambda=counter,m=2,N=i)
                    ave_of_all_exp_of_negative_rho_of_this_level=statistics.
 ↪mean(test)
                    occurence=0
                    tmp=[]
                    for node in test:
                        if node < ave_of_all_exp_of_negative_rho_of_this_level:
                            occurence+=1
                        else:
                            tmp.append(node)
                    print("N="+str(i))
                    print("occurence="+str(occurence))
                    if len(tmp)!=0:
                        ␣
 ↪ave_of_all_large_exp_of_negative_rho_of_this_level=statistics.mean(tmp)
                        rho.
 ↪append(ave_of_all_large_exp_of_negative_rho_of_this_level)
                        ␣
 ↪print("ave_of_all_large_exp_of_negative_rho_of_this_level:
 ↪"+str(ave_of_all_large_exp_of_negative_rho_of_this_level))
                        print("ave_of_all_exp_of_negative_rho_of_this_level:
 ↪"+str(ave_of_all_exp_of_negative_rho_of_this_level))
                        if ave_of_all_large_exp_of_negative_rho_of_this_level!
 ↪=0:
                            rho.
 ↪append(ave_of_all_large_exp_of_negative_rho_of_this_level/
 ↪ave_of_all_exp_of_negative_rho_of_this_level)
                            ␣
 ↪print("ave_of_all_short_exp_of_negative_rho_of_this_level/
 ↪ave_of_all_exp_of_negative_rho_of_this_level:
 ↪"+str(ave_of_all_large_exp_of_negative_rho_of_this_level/
 ↪ave_of_all_exp_of_negative_rho_of_this_level))
```

```
            except:
                print("---")
        if len(rho)>2:
            if rho[-1]>1:
                useful_example+=1

        counter+=increment
        print("counter:"+str(counter))
        print("useful_example:"+str(useful_example))

def examples_of_10000_with_t(initial, increment,t0):
    counter=initial
    useful_example=0
    while counter < initial+10000*increment:
        print("************"+str(counter)+"************")
        out=[]
        if check_T_generate_a_Schottky(Lambda=counter,m=2):
            try:
                rho=[]
                for i in range(15):
                    sum1=0
                    sum2=0
                    test=Exp_negative_Hyperbolic_Distance_Gamma0_with_t(Lambda=counter,m=2,N=i,t=t0)
                    #test=Exp_negative_Hyperbolic_Distance_Gamma0_with_t(Lambda=0.3,m=2,N=i,t=t0)
                    ave_of_all_exp_of_negative_rho_of_this_level=statistics.mean(test)
                    occurence=0
                    tmp=[]
                    for node in test:
                        if node < ave_of_all_exp_of_negative_rho_of_this_level:
                            occurence+=1
                        else:
                            tmp.append(node)
                    print("N="+str(i))
                    print("occurence="+str(occurence))
                    if len(tmp)!=0:
                        ave_of_all_large_exp_of_negative_rho_of_this_level=statistics.mean(tmp)
                        rho.append(ave_of_all_large_exp_of_negative_rho_of_this_level)
                        print("ave_of_all_large_exp_of_negative_rho_of_this_level:"+str(ave_of_all_large_exp_of_negative_rho_of_this_level))
```

```
                        print("ave_of_all_exp_of_negative_rho_of_this_level:
↪"+str(ave_of_all_exp_of_negative_rho_of_this_level))
                        if ave_of_all_large_exp_of_negative_rho_of_this_level!
↪=0:
                            rho.
↪append(ave_of_all_large_exp_of_negative_rho_of_this_level/
↪ave_of_all_exp_of_negative_rho_of_this_level)
                            ␣
↪print("ave_of_all_large_exp_of_negative_rho_of_this_level/
↪ave_of_all_exp_of_negative_rho_of_this_level:
↪"+str(ave_of_all_large_exp_of_negative_rho_of_this_level/
↪ave_of_all_exp_of_negative_rho_of_this_level))

            except:
                print("---")
        if len(rho)>2:
            if rho[-1]>1:
                useful_example+=1

        counter+=increment
        print("counter:"+str(counter))
        print("useful_example:"+str(useful_example))
# measuring hyperbolic distance
def Gamma(Lambda,m,N):

    T=operator_T(Lambda) # T is a 2 by 2 matrix.
    theta=float(math.pi/m)
    ID=[[[1, 0], [0, 0]], [[0, 0], [1, 0]]]
    R=operator_R(theta) # R is a 2 by 2 matrix.
    L=[T]
    tmp1=[]
    tmp2=[]
    model=[]
    component=[]

    nodes_in_DT=[ [[[1, 0], [0, 0]], [[0, 0], [1, 0]]]  ]
    j=1
    while j<=N:
        #N=1
        if j==1:
            z=L[0]
            tmp1=[]
            tmp2=[]
            for i in range(2*m-1):
                z=Cartesian_complex_matrix_multiplication(R,z)   #Tz(R,z)
                tmp1.append(z)
```

```
                if i!=m-1:
                    tmp2.append(z)
                else:
                    tmp2.append(L[0])
            L=[]
            for k in tmp2:
                L.append(k)
            nodes_in_DT=[Cartesian_complex_matrix_multiplication(T,ID)]

        #N>1
        else:
            nodes_in_DT=[]
            tmp1=[]
            tmp2=[]
            tmp3=[]

            for k in L:
                z=Cartesian_complex_matrix_multiplication(T,k)       #Tz(T,k)
                nodes_in_DT.append(z)
                tmp1.append(z)
                tmp2.append(z)







            L=[]
            for i in range(2*m-1):
                if i!=m-1:
                    for k in tmp1:
                        tmp3.
↪append(Cartesian_complex_matrix_multiplication(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                        L.append(k)
                    tmp3=[]
                elif i==m-1:
                    for k in tmp1:
                        tmp3.
↪append(Cartesian_complex_matrix_multiplication(R,k))
                    tmp1=[]
                    for k in tmp3:
                        tmp1.append(k)
                    tmp3=[]
                    for k in tmp2:
```

```
                        L.append(k)

        j+=1
        Out=[]
    index=0
    temp0=[]
    tmp1=[]
    for k in L:
        z=Cartesian_complex_matrix_multiplication(T,k)       #Tz(T,k)
        tmp1.append(z)
    N=int(math.log(len(tmp1),(2*m-1)))
    print("N:"+str(N))
    initial_index=(((2*m)-1)**(N-1))*(m+1)
    tindex=initial_index
    #print(len(tmp1))
    cindex=0
    counter=0
    for k in tmp1:

        if counter==(2*m-1):
            counter=0
            cindex+=1

        temp=[]
        component=[]
        component.append(initial_index)
        component.append(cindex)
        initial_index+=1
        initial_index=initial_index%((2*m)*(2*m-1)**(N-1))
        temp.append(k)
        temp.append(component)
        temp0.append(temp)
        counter+=1
    model.append(temp0)
    initial_index=tindex+(2*m-1)**(N-1)
    initial_index=initial_index%((2*m)*(2*m-1)**(N-1))
    tindex+=1
    tindex=tindex%((2*m)*(2*m-1)**(N-1))


    temp=[]
    temp1=[]
    temp2=[]


    print("tmp1"+str(len(tmp1)))
    for i in range(2*m-1):
```

```
        temp2=[]
        for k in tmp1:
            temp=[]
            temp0=[]
            if counter==(2*m-1):
                counter=0
                cindex+=1
            z=Cartesian_complex_matrix_multiplication(R,k)
            temp2.append(z)
            temp.append(z)
            component=[]
            component.append(initial_index)
            component.append(cindex)
            temp.append(component)
            initial_index+=1
            initial_index=initial_index%((2*m)*(2*m-1)**(N-1))
            temp0.append(temp)

            counter+=1
            model.append(temp0)
        tmp1=[]
        for k in temp2:
            tmp1.append(k)
        temp2=[]


        initial_index=tindex+(2*m-1)**(N-1)
        initial_index=initial_index%((2*m)*(2*m-1)**(N-1))
        tindex+=1
        tindex=tindex%((2*m)*(2*m-1)**(N-1))
    #for k in model:
    #    tmp1=[]
    #    component=[]
    #    for key in k:
    #
    #        print("====")
    #        print(key)
    #    print("------")


    #Hyperbolic_distance=[]
    #for k in nodes_in_DT:
    #    Hyperbolic_distance.append(Cartesian_radial_hyperbolic_distance(k))
    return model
```

```
def non_normalized_derivative(T,z):

    #T=[[[a,b],[c,-d]], [[c,d],[a,-b]]]
    #z=[x,y]
    #a1=T[0][0][0]
    #a2=T[0][0][1]
    #b1=T[0][1][0]
    #b2=T[0][1][1]
    #c1=T[1][0][0]
    #c2=T[1][0][1]
    #d1=T[1][1][0]
    #d2=T[1][1][1]
    a=T[0][0]
    b=T[0][1]
    c=T[1][0]
    d=T[1][1]

   ␣
 ↪#NR=d1**2-d2**2+2*c1*d1*x-2*c2*d2*x+c1**2*x**2-c2**2*x**2-2*c2*d1*y-2*c1*d2*y-4*c1*c2*x*y-c
   ␣
 ↪#NI=2*d1*d2+2*c2*d1*x+2*c1*d2*x+2*c1*c2*x**2+2*c1*d1*y-2*c2*d2*y+2*c1**2*x*y-2*c2**2*x*y-2*
    #DR=-b1*c1+b2*c2+a1*d1-a2*d2
    #DI=-b2*c1-b1*c2+a2*d1+a1*d2
    #N1=NR*DR+NI*DI
    #N2=NI*DR-NR*DI
    #D=DR**2+DI**2
   ␣
 ↪N=Cartesian_complex_add(Cartesian_complex_mul(a,d),Cartesian_complex_scalar_mul(-1,Cartesia
   ␣
 ↪D=D=Cartesian_complex_mul(Cartesian_complex_add(Cartesian_complex_mul(c,z),d),Cartesian_com

    if Cartesian_complex_modulus(D)!=0:
        return Cartesian_complex_divide(N,D)
    else:
        return "null"


def derivatives(model):
    output=[]
    for i in model:
        for j in i:
            tmp=[]
            z=Tz(j[0],[0,-1])
            D_of_T=derivative(j[0],z)
            if D_of_T!="null":
                Tij=float(1/Cartesian_complex_modulus(D_of_T))
            else:
```

```
                Tij=0
            tmp.append(Tij)
            tmp.append(j[1])
            output.append(tmp)
    return output
```

[387]:
```
# Generate x_j
def Generate_xj(M,x_1):
    #M=number of disks in the first level
    #x_1=[1,0]
    l=[]

    k=2
    theta=float(2*math.pi/M)
    mul=Polar_complex_complex_to_Cartesian([1,theta])
    #M=3
    tmp=x_1
    l.append(tmp)
    while k<=M:
        tmp=Cartesian_complex_mul(mul,tmp)
        l.append(tmp)
        k+=1
    return l
# Compute y_ij
def inverse_f1(R,z,q):
    D=Cartesian_complex_add(z,Cartesian_complex_scalar_mul(-1,q))
    numb2=Cartesian_complex_divide([R**2,0],D)
    return Cartesian_complex_add(Cartesian_complex_complex_conjugate(q),numb2)
def inverse_f2(R,z,q):
    D=Cartesian_complex_add(z,Cartesian_complex_scalar_mul(-1,q))
    numb2=Cartesian_complex_divide([R**2,0],D)
    H2=Cartesian_complex_divide([1,-math.sqrt(3)],[1,math.sqrt(3)])
    H2bar=Cartesian_complex_complex_conjugate(H2)
    return␣
 ↪Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_complex_conjugate(q),numb2
def inverse_f3(R,z,q):
    D=Cartesian_complex_add(z,Cartesian_complex_scalar_mul(-1,q))
    numb2=Cartesian_complex_divide([R**2,0],D)
    H3=Cartesian_complex_divide([-1,-math.sqrt(3)],[-1,math.sqrt(3)])
    H3bar=Cartesian_complex_complex_conjugate(H3)
    return␣
 ↪Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_complex_conjugate(q),numb2
```

[519]:
```
import numpy as np
from numpy.linalg import eig
```

```python
def Symmetric_pairs_of_pants(angle,tinitial,incre):
    Theta=angle#math.pi-angle#0.5*angle
    R=math.tan((Theta/2)) #math.sqrt(2-2*math.cos(Theta))
    tmpl=Generate_xj(M=3,x_1=[math.sqrt(1+R**2),0])
    tmp2=Generate_xj(M=3,x_1=[0.7320628133,0])

    t11=0
    t22=0
    t33=0
    a=tmpl[0]
    b=Cartesian_complex_add([R**2,0],Cartesian_complex_scalar_mul(-1,Cartesian_complex_mul(tmpl
    Cartesian_complex_complex_conjugate(tmpl[0]))))
    c=[1,0]
    d=Cartesian_complex_scalar_mul(-1,Cartesian_complex_complex_conjugate(tmpl[0]))
    T=[[a,b],[c,d]]
    y12=Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_mul(a,tmp2[1]),b),
                                                       Cartesian_complex_add(Cartesian_complex_mul(c,tmp2[1]),d))
    y13=Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_mul(a,tmp2[2]),b),
                                                       Cartesian_complex_add(Cartesian_complex_mul(c,tmp2[2]),d))
    t12=float(1/
    Cartesian_complex_modulus(non_normalized_derivative([[tmpl[0],b],[[1,0],
        Cartesian_complex_scalar_mul(-1,Cartesian_complex_complex_conjugate(tmpl[0]))]],y12)))
    t13=float(1/
    Cartesian_complex_modulus(non_normalized_derivative([[tmpl[0],b],[[1,0],
        Cartesian_complex_scalar_mul(-1,Cartesian_complex_complex_conjugate(tmpl[0]))]],y13)))
    H2=Cartesian_complex_divide([1,-math.sqrt(3)],[1,math.sqrt(3)])
    H2bar=Cartesian_complex_complex_conjugate(H2)
    q2=tmpl[1]
    a=Cartesian_complex_mul(q2,H2bar)
    b=Cartesian_complex_add([R**2,0],Cartesian_complex_scalar_mul(-1,Cartesian_complex_mul(tmpl
    Cartesian_complex_complex_conjugate(tmpl[1]))))
    c=H2bar
    d=Cartesian_complex_scalar_mul(-1,Cartesian_complex_complex_conjugate(q2))
    T=[[a,b],[c,d]]
    y21=Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_mul(a,tmp2[0]),b),
```

```
                                 ␣
↪Cartesian_complex_add(Cartesian_complex_mul(c,tmp2[0]),d))
  ␣
↪y23=Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_mul(a,tmp2[2]),b),
                                 ␣
↪Cartesian_complex_add(Cartesian_complex_mul(c,tmp2[2]),d))
  t21=float(1/
↪Cartesian_complex_modulus(non_normalized_derivative([[a,b],[c,d]],y21)))
  t23=float(1/
↪Cartesian_complex_modulus(non_normalized_derivative([[a,b],[c,d]],y23)))
  H3=Cartesian_complex_divide([-1,-math.sqrt(3)],[-1,math.sqrt(3)])
  H3bar=Cartesian_complex_complex_conjugate(H3)
  q3=tmpl[2]
  a=Cartesian_complex_mul(q3,H3bar)
  ␣
↪b=Cartesian_complex_add([R**2,0],Cartesian_complex_scalar_mul(-1,Cartesian_complex_mul(tmpl
  Cartesian_complex_complex_conjugate(tmpl[2]))))
  c=H3bar
  d=Cartesian_complex_scalar_mul(-1,Cartesian_complex_complex_conjugate(q3))
  T=[[a,b],[c,d]]
  ␣
↪y31=Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_mul(a,tmp2[0]),b),
                                 ␣
↪Cartesian_complex_add(Cartesian_complex_mul(c,tmp2[0]),d))
  ␣
↪y32=Cartesian_complex_divide(Cartesian_complex_add(Cartesian_complex_mul(a,tmp2[1]),b),
                                 ␣
↪Cartesian_complex_add(Cartesian_complex_mul(c,tmp2[1]),d))
  t31=float(1/
↪Cartesian_complex_modulus(non_normalized_derivative([[a,b],[c,d]],y31)))
  t32=float(1/
↪Cartesian_complex_modulus(non_normalized_derivative([[a,b],[c,d]],y32)))
  #print(t11)
  #print(t12)
  #print(t13)
  #print(t21)
  #print(t22)
  #print(t23)
  #print(t31)
  #print(t32)
  #print(t33)
  TIJ=[[t11,t12,t13],[t21,t22,t23],[t31,t32,t33]]
  #print(TIJ)
  flag=1
  flag2=0
  while flag==1:
```

```python
        col=[]
        TIJ_l=[]
        for k in TIJ:
            col=[]
            for i in k:
                val=i**tinitial
                col.append(val)
            TIJ_l.append(col)
        a = np.array(TIJ_l)
        w,v=eig(a)
        M_l=-10000000000
        for k in w:
            if k>M_l:
                M_l=k
        #print(M_l)
        #print("tinitial:"+str(tinitial))
        if M_l>1 and flag2==0:
            flag2=1
        if flag2==1 and M_l<1:
            flag=0
        if M_l<1 and flag2==0:
            flag2=-1
        if flag2==-1 and M_l>1:
            flag=0
        tinitial+=incre
    print("delta:"+str(tinitial))
    #print("Max eigenvalue:"+str(M_l.real))
    return tinitial
```

[520]:
```python
import matplotlib.pyplot as plt
mlist=[]
for angle in range(1,121):
    print("angle:"+str(angle))
    result=Symmetric_pairs_of_pants(angle=float(angle/180)*math.pi,tinitial=0.
 ↪000,incre=0.00001)
    mlist.append(result)
Xlist=[]
angle=1
for k in mlist:
    Xlist.append(angle)
    angle+=1
plt.plot(Xlist, mlist)
```

```
angle:1
delta:0.0673000000000066
angle:2
delta:0.07776000000000255
angle:3
```

```
delta:0.08553999999999953
angle:4
delta:0.092079999999997
angle:5
delta:0.09787999999999475
angle:6
delta:0.10318999999999269
angle:7
delta:0.10815999999999076
angle:8
delta:0.11285999999998894
angle:9
delta:0.1173599999999872
angle:10
delta:0.12170999999998551
angle:11
delta:0.12591999999998515
angle:12
delta:0.13002999999998927
angle:13
delta:0.1340599999999933
angle:14
delta:0.13801999999999726
angle:15
delta:0.14192000000000116
angle:16
delta:0.14578000000000502
angle:17
delta:0.14960000000000884
angle:18
delta:0.15339000000001263
angle:19
delta:0.1571600000000164
angle:20
delta:0.16090000000002014
angle:21
delta:0.16464000000002388
angle:22
delta:0.1683700000000276
angle:23
delta:0.17209000000003133
angle:24
delta:0.17582000000003506
angle:25
delta:0.1795500000000388
angle:26
delta:0.18328000000004252
angle:27
```

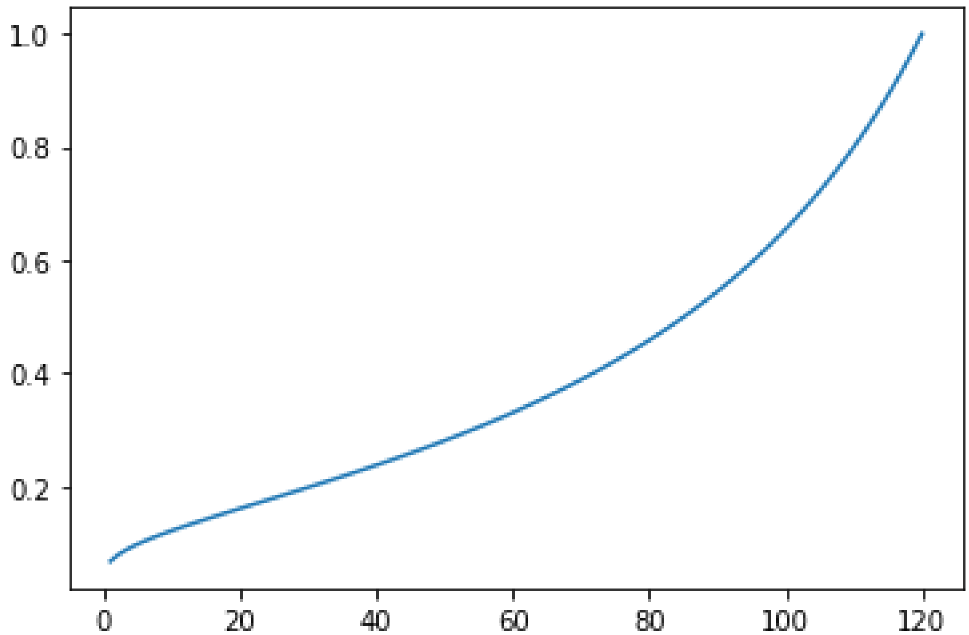
```
delta:0.18703000000004627
angle:28
delta:0.19078000000005002
angle:29
delta:0.1945500000000538
angle:30
delta:0.19834000000005758
angle:31
delta:0.2021500000000614
angle:32
delta:0.20598000000006522
angle:33
delta:0.20984000000006908
angle:34
delta:0.21372000000007296
angle:35
delta:0.21763000000007687
angle:36
delta:0.2215700000000808
angle:37
delta:0.2255500000000848
angle:38
delta:0.2295600000000888
angle:39
delta:0.23360000000009284
angle:40
delta:0.23769000000009693
angle:41
delta:0.24181000000010106
angle:42
delta:0.24598000000010523
angle:43
delta:0.25019000000010944
angle:44
delta:0.2544400000001137
angle:45
delta:0.258750000000118
angle:46
delta:0.26311000000012236
angle:47
delta:0.26751000000012676
angle:48
delta:0.2719700000001312
angle:49
delta:0.27649000000013574
angle:50
delta:0.2810700000001403
angle:51
```

```
delta:0.28570000000014495
angle:52
delta:0.29040000000014965
angle:53
delta:0.2951700000001544
angle:54
delta:0.30000000000015925
angle:55
delta:0.30490000000016415
angle:56
delta:0.3098700000001691
angle:57
delta:0.31491000000017416
angle:58
delta:0.3200300000001793
angle:59
delta:0.3252400000001845
angle:60
delta:0.3305200000001898
angle:61
delta:0.33589000000019514
angle:62
delta:0.3413400000002006
angle:63
delta:0.34689000000020614
angle:64
delta:0.3525200000002118
angle:65
delta:0.3582600000002175
angle:66
delta:0.36409000000022335
angle:67
delta:0.3700300000002293
angle:68
delta:0.3760700000002353
angle:69
delta:0.3822200000002415
angle:70
delta:0.38848000000024774
angle:71
delta:0.39487000000025413
angle:72
delta:0.40137000000026063
angle:73
delta:0.40799000000026725
angle:74
delta:0.414750000000274
angle:75
```

```
delta:0.4216400000002809
angle:76
delta:0.42867000000028793
angle:77
delta:0.4358400000002951
angle:78
delta:0.4431500000003024
angle:79
delta:0.4506200000003099
angle:80
delta:0.4582500000003175
angle:81
delta:0.4660400000003253
angle:82
delta:0.47400000000033327
angle:83
delta:0.4821400000003414
angle:84
delta:0.4904600000003497
angle:85
delta:0.49896000000035823
angle:86
delta:0.5076600000003244
angle:87
delta:0.5165600000002839
angle:88
delta:0.5256700000002424
angle:89
delta:0.5350000000002
angle:90
delta:0.5445500000001565
angle:91
delta:0.554340000000112
angle:92
delta:0.5643700000000663
angle:93
delta:0.5746600000000195
angle:94
delta:0.5851999999999715
angle:95
delta:0.5960199999999223
angle:96
delta:0.6071199999998718
angle:97
delta:0.6185099999998199
angle:98
delta:0.6302099999997667
angle:99
```

```
delta:0.642229999999712
angle:100
delta:0.6545899999996557
angle:101
delta:0.667279999999598
angle:102
delta:0.6803399999995385
angle:103
delta:0.6937799999994774
angle:104
delta:0.7076099999994144
angle:105
delta:0.7218499999993496
angle:106
delta:0.7365099999992829
angle:107
delta:0.7516299999992141
angle:108
delta:0.7672099999991432
angle:109
delta:0.78327999999907
angle:110
delta:0.7998599999989946
angle:111
delta:0.8169799999989167
angle:112
delta:0.8346599999988362
angle:113
delta:0.8529299999987531
angle:114
delta:0.8718199999986671
angle:115
delta:0.8913599999985782
angle:116
delta:0.9115799999984862
angle:117
delta:0.9325099999983909
angle:118
delta:0.9541999999982922
angle:119
delta:0.9766799999981899
angle:120
delta:1.0000099999980838
```

[520]:
```
[<matplotlib.lines.Line2D at 0x1090d34f0>]
```



1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100
120


```
[ ]:
```