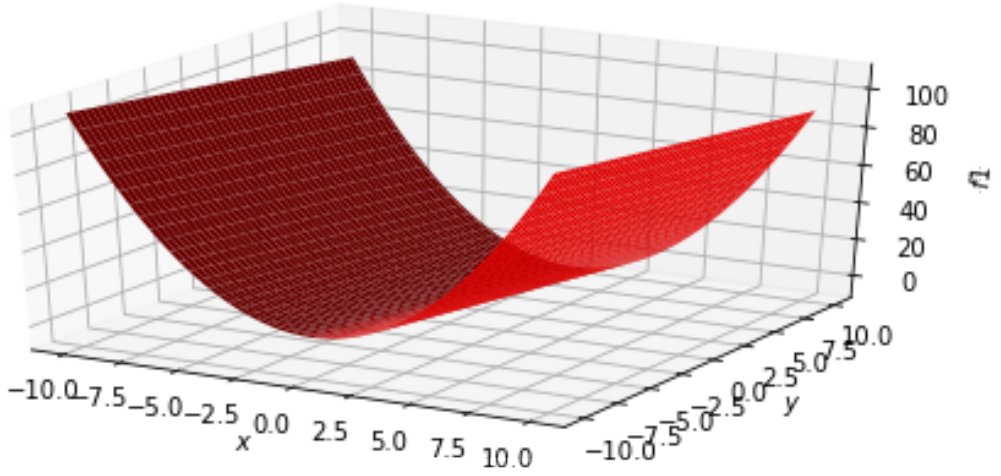
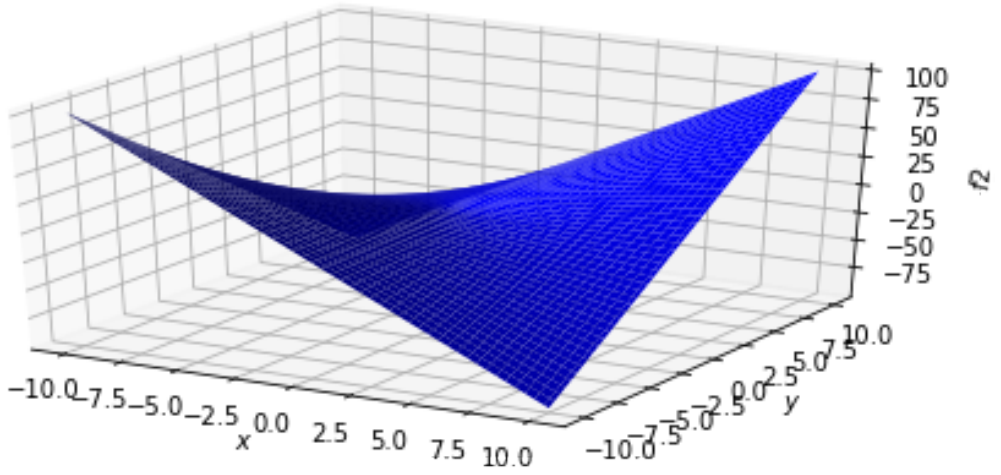
pathtracking3

February 27, 2020

[65]:
```
import matplotlib.pyplot as plt
import numpy as np
import newton
from cmath import *
from math import *
from mpl_toolkits.mplot3d import Axes3D

#we are going to look at a system of polynomials now.

def f(x,y):
    return np.array([x**2 -y,x*y+1])

X=np.linspace(-10,10,num=100)
Y=np.linspace(-10,10,num=100)
u,v=np.meshgrid(X,Y)
f1=f(u,v)[0]
f2=f(u,v)[1]

fig = plt.figure(figsize=(8,8))
ax1 = fig.add_subplot(211, projection='3d')
ax2 = fig.add_subplot(212, projection='3d')
surf1=ax1.plot_surface(u,v,f1,color="r")
surf2=ax2.plot_surface(u,v,f2,color="b")
ax1.set_xlabel("$x$")
ax2.set_xlabel("$x$")
ax1.set_ylabel("$y$")
ax2.set_ylabel("$y$")
ax1.set_zlabel("$f 1$")
ax2.set_zlabel("$f 2$")
plt.show()
```






```
[ 5 13]
```

[97]:
```
# Path tracking section
gamma=0.44+0.77j

def g(x,y):
    return np.array([x**2-1,y**2-1]) #sp g(x,y)=0 has solutions␣
↪(1,1),(1,-1),(-1,1) and (-1,-1).

def Jf(x,y):
    # Jacobian matrix of f wrt (x,y)
    return np.array([[2*x,-1],[y,x]])

def Jg(x,y):
```

```
    # Jacobian matrix of g wrt (x,y)
    return np.array([[2*x,0],[0,2*y]])

def H(x,y,t):
    # Homotopy between f and g
    return gamma*t*g(x,y)+(1-t)*f(x,y) #0<=t<=1

def JH(x,y,t):
    # Jacobian of H wrt (x,y)
    return gamma*t*Jg(x,y)+(1-t)*Jf(x,y)

def Dt_H(x,y):
    # Derivative of H wrt t
    return gamma*g(x,y)-f(x,y)

def euler2D(x,y,t,step,func):
    # 2D explicit euler iteration
    # x,y: 2D initial spacial values
    # t: intial time
    # func: function returning 2D array satisfying d(x,y)/dt=func(x,y)
    return np.array([x,y])-step*func(x,y,t)

def F(x,y,t):
    return -np.dot(np.linalg.inv(JH(x,y,t)), Dt_H(x,y))

def iterate2D(x0,y0,t0,dt):
    # single path tracking iteration
    # path tracking: t=1 -> t=0 (dt<0)
    z0=euler2D(x0,y0,t0,dt,F)
    z1=z0-np.dot(np.linalg.inv(JH(z0[0],z0[1],t0+dt)),H(z0[0],z0[1],t0))
    z2=z1-np.dot(np.linalg.inv(JH(z1[0],z1[1],t0+2*dt)),H(z1[0],z1[1],t0+dt))
    return z2
X1=complex(1,0)
Y1=complex(1,0)

X2=complex(-1,0)
Y2=complex(1,0)

X3=complex(-1,0)
Y3=complex(-1,0)
T=1
N=100
for i in range(0,N):
    s1=iterate2D(X1,Y1,T,dt=-1/N)
    s2=iterate2D(X2,Y2,T,dt=-1/N)
    s3=iterate2D(X3,Y3,T,dt=-1/N)
    X1=s1[0]
```

```
    Y1=s1[1]
    X2=s2[0]
    Y2=s2[1]
    X3=s3[0]
    Y3=s3[1]
    T-=1/N
print(X1,Y1) #sol1
print(X2,Y2) #sol2
print(X3,Y3) #sol3
```

```
(0.5000094435171657+0.8658651385239723j)
(-0.4999003700860926+0.8660816417632938j)
(-1+0j) (1+0j)
(0.50004363403548-0.8659579019697261j) (-0.5000518566508412-0.8660039317402998j)
```

[ ]:

[72]:
```
# exact solutions comparison
sol1=[complex(0.5,0.5*sqrt(3)),complex(-0.5,0.5*sqrt(3))]
sol2=[-1,1]
sol3=[complex(0.5,-0.5*sqrt(3)),complex(-0.5,-0.5*sqrt(3))]
print("sol1: ",sol1)
print("\nsol2: ",sol2)
print("\nsol3: ",sol3)
#As we can see, it works! And has generated all the solutions.
```

```
sol1:  [(0.5+0.8660254037844386j), (-0.5+0.8660254037844386j)]

sol2:  [-1, 1]

sol3:  [(0.5-0.8660254037844386j), (-0.5-0.8660254037844386j)]
```