Ay 190 Final Project

The Diffusion Equation

March 25, 2013

1 The Roche potential

The Roche potential is given by

$$V(x,y) = -\frac{GM_1}{r_1} - \frac{GM_2}{r_2} - \omega^2\frac{x^2+y^2}{2}, \tag{1}$$

where $r_1 = \sqrt{(x-a)^2+y^2}$ and $r_2 = \sqrt{(x+b)^2+y^2}$. An example potential is shown in Figure 1.

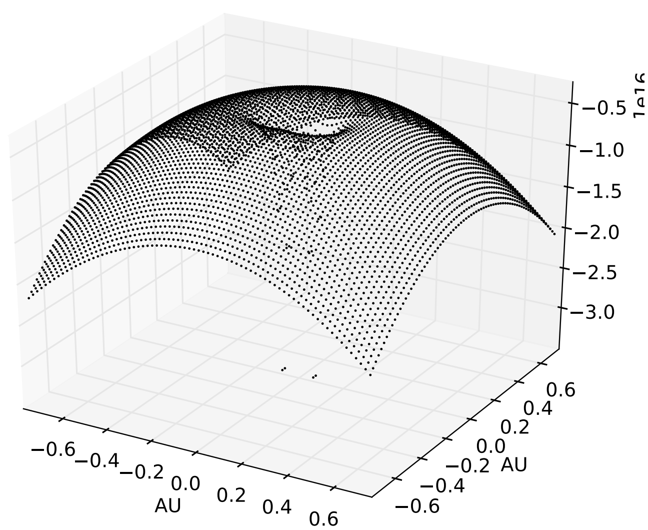


Figure 1: Equipotential surfaces for Cyg X-1, assuming $M_* = 40\ M_\odot$ and $M_{BH} = 20 M_\odot$, that their separation is 0.2 AU and have a period of 5.6 days.

2 Roche surfaces

2D equipotential Roche surfaces were computed by for various potential V by finding the root of Equation 1 multiplied by r_1 and r_2 using the bisection method for an array of x values. To obtain 3D surfaces, the 2D surfaces were rotated about the x-axis, about which the potential is symmetric. The full code used to compute the surfaces are shown in Appendix A.

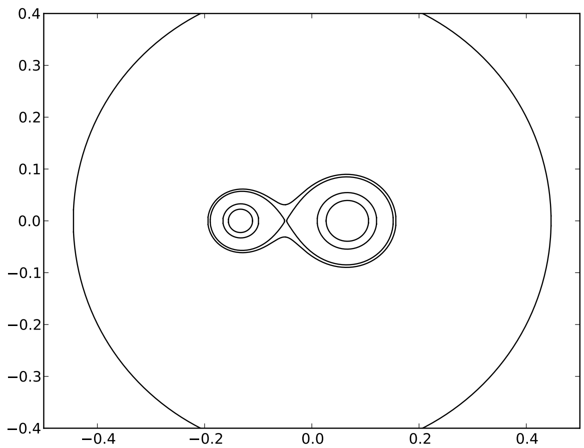
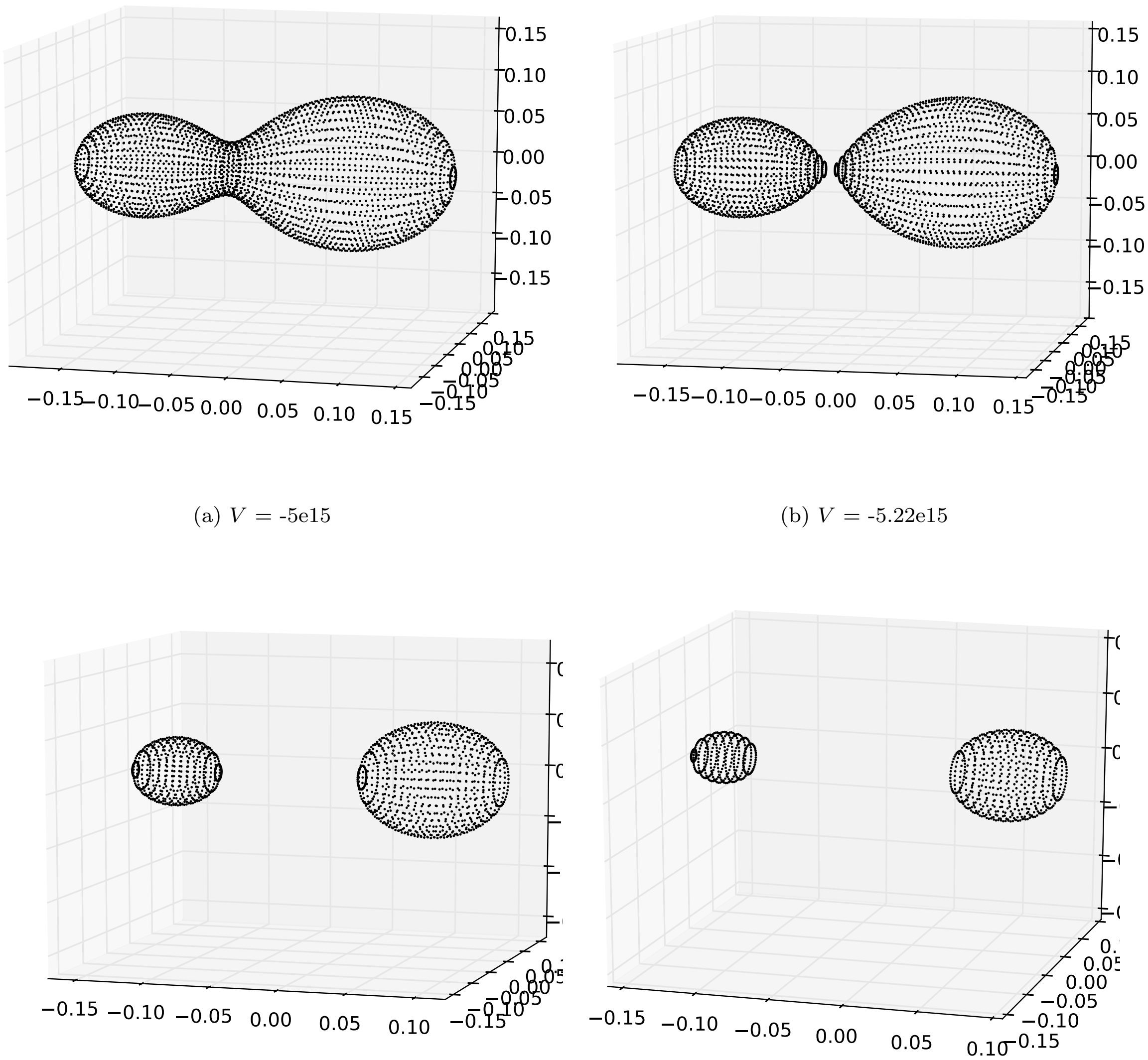


Figure 2: Roche surfaces of the Cyg X-1 system in 2D. From inside to out, the potentials of the surfaces are -1e-16, -7.5e15, -5.22e15 (the surface with the Lagrange point L1), -5e15, and -4.99999999999e+15.

(Please see next page for 3D surfaces.)



(a) V = -5e15

(b) V = -5.22e15

(c) V = -7.5e15

(d) V = -1e16

Figure 3: 3D Roche surfaces for the Cyg X-1 system.

Appendices

The Python modules and scripts used in this assignment include

A Computing Roche Surfaces: `roche.py` (see pages 4-6)

B Root-finding Routines: `findroot.py` (see pages 6-9)

A Computing Roche Surfaces

```
# roche.py
# created by stacy kim

from numpy import *
from findroot import *
from matplotlib.pyplot import *
from mpl_toolkits.mplot3d import Axes3D


G = 6.6733e-8     # gravitational constant
MSUN = 1.9891e33  # mass of the sun (g)
AU = 1.49598e13
DAYS = 3600*24   # seconds in a day

err=2e26

M1 = 40*MSUN # mass of primary
M2 = 20*MSUN # mass of secondary

sep=0.2*AU         # distance between primary and secondary
a = M2*sep/(M1+M2) # location of primary
b = M1*sep/(M1+M2) # location of secondary
omega = 2*pi / (5.5998 * DAYS) # angular velocity of orbiting bodies

V = lambda x2, y2: -(G*M1/sqrt((sqrt(x2)-a)**2 + y2)
                    + G*M2/sqrt((sqrt(x2)+b)**2 + y2)
                    + omega**2 * (x2 + y2)/2.)


# ================================================================================
# PLOT POTENTIAL
fig = figure()
ax = fig.add_subplot(111,projection='3d')
xlabel('AU')
ylabel('AU')

v=[]
mi=-0.75
ma=0.75
st=0.02
```

```
for x in arange(mi*AU,ma*AU,st*AU):
    for y in arange(mi*AU,ma*AU,st*AU):
        v+=[V(x*x,y*y)]
n=len(arange(mi,ma,st))
y=concatenate([arange(mi,ma,st)]*n)
x=concatenate([ones(n)*y[i] for i in range(n)])
ax.plot(x,y,v,'ko',ms=1)
savefig('potential.pdf')
show()


# ============================================================================
# PLOT ROCHE SURFACES

def f(x,y,v):
    r1 = sqrt( (x-a)**2 + y**2 )
    r2 = sqrt( (x+b)**2 + y**2 )
    x3 = omega**2 * r1 * r2 * (x**2 + y**2)/2.
    return r1*r2*v + G*(M1*r2 + M2*r1) + x3

mid=-0.0488
err = 2e26
surfaces=[]
for v in [-5e15+1e4,-5e15,-5.22e15,-7.5e15,-1e16]:
    print v
    st = 0.005 #if v != -5.22e15 else 0.0001
    if v <= -5*10**15: mi,ma = -0.25,0.25
    else:              mi,ma = -0.60,0.60

    # compute surface for negative x
    xsurface=[]
    ysurface=[]
    for x in arange(mi*AU,mid*AU,st*AU):
        ff = lambda y: f(x,y,v)
        try:
            yy,cnt=bisection(0,ma*AU,ff,err)
            xsurface+=[x]
            ysurface+=[yy]
        except ValueError:  pass

    surfaces+=[[array(xsurface)/AU,array(ysurface)/AU]]

    # calculate surface for positive x separately
    xsurface=[]
    ysurface=[]
    for x in arange(mid*AU,ma*AU,st*AU):
        ff = lambda y: f(x,y,v)
        try:
            yy,cnt=bisection(0,ma*AU,ff,err)
            xsurface+=[x]
            ysurface+=[yy]
```

```
        except ValueError:  pass

    surfaces+=[[array(xsurface)/AU,array(ysurface)/AU]]


# plot 2D surfaces
for i,surface in enumerate(surfaces):
    plot(surface[0], surface[1],'k')
    plot(surface[0],-surface[1],'k')

    # connect edges of surfaces across y=0
    if not (i%2==1 and i<=3):
        plot( [surface[0][0]]*2, [surface[1][0],-surface[1][0]], 'k')
    if not (i%2==0 and i<=3):
        plot( [surface[0][-1]]*2, [surface[1][-1],-surface[1][-1]], 'k')

ylim([-0.4,0.4])
savefig('roche_surfaces2D.pdf')
show()


# plot 3D surfaces
fig=figure()
ax = fig.add_subplot(111,projection='3d')
n=len(arange(0,pi,pi/20))

yz = concatenate([surfaces[0][1],-surfaces[0][1]])
z = concatenate([yz*sin(th) for th in arange(0,pi,pi/20)])
y = concatenate([yz*cos(th) for th in arange(0,pi,pi/20)])
x  = concatenate([surfaces[0][0]]*n*2)
plot(x,y,z,'ko',ms=1)

yz = concatenate([surfaces[1][1],-surfaces[1][1]])
z = concatenate([yz*sin(th) for th in arange(0,pi,pi/20)])
y = concatenate([yz*cos(th) for th in arange(0,pi,pi/20)])
x  = concatenate([surfaces[1][0]]*n*2)
plot(x,y,z,'ko',ms=1)

ylim(xlim())
ax.set_zlim(xlim())

show()
```

B Root-finding Routines

```
# findroot.py
# created 11/17/11 by stacy kim
#
# Contains a suite of routines to find roots that implement the methods
```

```
#    Newton-Raphson
#    secant
#    bisection.
#
#
# modified 1/26/13: removed support to output to file, instead returning array
# modified 1/28/13 to return number of iterations required to find solution
# modified 1/29/13: added method to find all rational roots to a polynomial


import sys
import math


# FILE OUTPUT ROUTINES ------------------------------------------------------------

files=None

def set_output(h,c,fns):
    """
    Enables output to files and sets file names for all 3 methods to those
    given in the array fns, in the order bisection, newton_raphson, secant.
    """
    global files

    # Open output files and write headers
    files=[0]*3

    for i in range(3):
        files[i]=open(fns[i],'w')
        files[i].write(fns[i])
        files[i].write('\nh={0},c={1},err=1e-4\n{2}{3}\n'.format(h,c,'x'.rjust(19),'precision'.rjust(19


def close_files():
    """Closes output files, if any."""
    if (files==None): return

    print "Finished writing output files."
    for i in range(3):
        files[i].close()


# ROOT-FINDING ROUTINES ----------------------------------------------------------

def bisection(x1,x2,f,err):

    if (f(x1)*f(x2) > 0):
        raise ValueError('f(x1) and f(x2) must have opposite signs!\n'
                         '   f({0})={1},  f({2})={3}'.format(x1,f(x1),x2,f(x2)))
```

```
    if (x2==min(x1,x2)): x1,x2 = x2,x1

    count=0
    x=(x1+x2)/2.0
    while(abs(f(x)) > err):
        if (f(x)*f(x1)>0): x1=x
        else: x2=x
        x=(x1+x2)/2.0

        count+=1
        if (count>=990):
            print 'f({0})={1}, f({2})={3}'.format(x1,f(x1),x2,f(x2))
            if (count==1000):
                print 'Failed to converge after 1000 iterations.'
                sys.exit()

    return x,count


def newton_raphson(x,f,df,err):
    count=0
    while(abs(f(x)) > err):
        x=x-f(x)/df(x)
        count+=1
    return x,abs(f(x)/df(x)),count


def secant(x2,x,f,err):
    count=0
    while(abs(f(x)) > err):
        x1=x2
        x2=x
        x=x2-f(x2)*(x2-x1)/(f(x2)-f(x1))
        count+=1
        if count % 10000 == 0: print count,':',x1,x2,x,f(x)

    df=(f(x2)-f(x))/(x2-x)
    return x,abs(f(x)/df),count


def find_all_roots(coeff0):
    """
    Computes all integer or rational roots of the given function
    f = sum([coeff[i]*x**i for i=0..len(coeff)]).
    """
    coeff=coeff0

    roots=[]
    while len(roots) < len(coeff0)-1:
        # find zero of polynomial
        f  = lambda x: sum([coeff[i] * x**i for i in range(len(coeff))])
```

```
    root,err,niter=secant(0.0,1.0,f,1e-15)
    roots.append(root)

    #synthetic division
    a=coeff[-1]
    for i in range(len(coeff)-2,-1,-1): coeff[i]=a=coeff[i]+a*root
    if not (abs(coeff[0]) <= 1e-10):
        print 'found false root x =',root,'! (',abs(coeff[0]),')'
        sys.exit()
    coeff=coeff[1:]

return roots
```