Distance of closest approach of two arbitrary hard ellipsoids

Xiaoyu Zheng
Department of Mathematical Sciences, Kent State University, Kent, Ohio, 44224

Wilder Iglesias and Peter Palffy-Muhoray*
Liquid Crystal Institute, Kent State University, Kent, Ohio, 44224

October 14, 2008

Abstract

The distance of closest approach of particles with hard cores is a key parameter in statistical theories and computer simulations of liquid crystals and colloidal systems. In this paper, we provide a new algorithm to calculate the distance of closest approach of two ellipsoids of arbitrary shape and orientation. This algorithm is based on our previous analytic result for the distance of closest approach of 2D ellipses. The method consists of determining the intersection of the ellipsoids with the plane containing the line joining their centers and rotating the plane. The distance of closest approach of the two ellipses formed by the intersection is a periodic function of the plane orientation, whose maximum corresponds to the distance of closest approach of the two ellipsoids.

1 Introduction

The distance of closest approach of particles with hard cores is a key parameter in statistical theories and computer simulations of liquid crystals and colloidal systems. Although ellipsoids are the simplest non-spherical shapes, no analytic solution exists for the distance of their closest approach. For this reason, in liquid crystal theories, spherocylindrical hard cores are often used, whose excluded volume can be exactly calculated [1]. In computer simulations of ellipsoids, overlap criteria are typically used [2, 3]. We have recently succeeded in obtaining a closed form analytic expression for the distance of closest approach of two hard ellipses of arbitrary size and eccentricity [4]. Here we present an algorithm for finding the closest approach of two ellipsoids based on this analytic result. This algorithm may be useful for calculating excluded volumes and related quantities, such as elastic constants, for liquid crystals and colloids, and it may also provide an overlap criterion for ellipsoids in computer simulations.

2 Description and solution of the problem

Consider two ellipsoids, each with a given shape and orientation, whose centers are on a line with given direction. We wish to determine the distance between centers when the ellipsoids are in point contact externally. This distance of closest approach is a function of the shapes of the ellipsoids and their orientation. There is no analytic solution for this problem, since solving for the distance requires the solution of a sixth order polynomial equation [4]. Here we present an algorithm to determine this distance, based on our analytic results for the distance of closest approach of ellipses in 2D, which can be implemented numerically. Our algorithm consists of three steps.

1. Constructing a plane containing the line joining the centers of the two ellipsoids, and finding the equations of the ellipses formed by the intersection of this plane and the ellipsoids.

2. Determining the distance of closest approach of the ellipses; that is the distance between the centers of the ellipses when they are in point contact externally.

http://www.e-lc.org/docs/2008_10_12_23_11_56

3. Rotating the plane until the distance of closest approach of the ellipses is a maximum. The distance of closest approach of the ellipsoids is this maximum distance.

We detail steps 1 and 3 in the sections below. Step 2 is described in Refs. [4, 5].

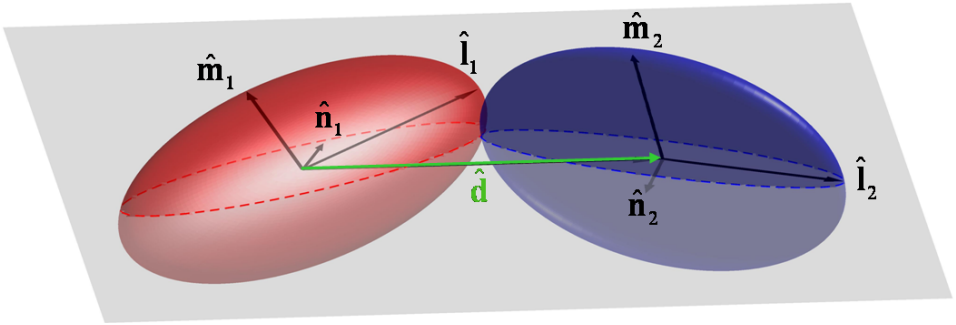

Figure 1: Two externally tangent ellipsoids intersecting with a plane containing the line joining the centers. The semiaxes of the ellipsoid on the left are 10, 4, 2, while those of the one on the right are 8, 6, 2. The vector joining the centers has direction $\langle 1, 0, 0\rangle$. The distance of their closest approach is 14.7163.

2.1 Step 1. Ellipses formed by the intersection of the ellipsoids and the plane

The shapes of the ellipsoids are specified by the lengths a, b, c of their principal axes; the orientations are given by the unit vectors $\hat{\mathbf{l}}$, $\hat{\mathbf{m}}$, $\hat{\mathbf{n}}$ along the principal axes. The equations of the ellipsoids have the form

$$\mathbf{r}\cdot\left(\frac{\hat{\mathbf{l}}\hat{\mathbf{l}}}{a^2}+\frac{\hat{\mathbf{m}}\hat{\mathbf{m}}}{b^2}+\frac{\hat{\mathbf{n}}\hat{\mathbf{n}}}{c^2}\right)\cdot\mathbf{r}=1. \tag{1}$$

We next assume that their centers are on a line whose direction is given by $\hat{\mathbf{d}}$.

We construct a plane which contains the line connecting the centers of the ellipsoids. The normal to the plane to is $\hat{\mathbf{p}}$, and we define $\hat{\mathbf{s}}=\hat{\mathbf{p}}\times\hat{\mathbf{d}}$.

Since we want to rotate the plane, we define the initial direction of the normal $\hat{\mathbf{p}}_0$ as

$$\hat{\mathbf{p}}_0=\frac{\hat{\mathbf{d}}\times\hat{\mathbf{l}}_1}{|\hat{\mathbf{d}}\times\hat{\mathbf{l}}_1|}. \tag{2}$$

If $\hat{\mathbf{p}}_0=0$, then

$$\hat{\mathbf{p}}_0=\frac{\hat{\mathbf{d}}\times\hat{\mathbf{m}}_1}{|\hat{\mathbf{d}}\times\hat{\mathbf{m}}_1|}. \tag{3}$$

We indicate rotation by the angle θ

$$\hat{\mathbf{p}}=(\cos\theta)\hat{\mathbf{p}}_0+(\sin\theta)(\hat{\mathbf{p}}_0\times\hat{\mathbf{d}}),\quad \theta\in[0,\pi). \tag{4}$$

The unit vectors along the principal axes of the ellipsoids can be expressed in term of these coordinates:

$$\hat{\mathbf{l}} = l_x\hat{\mathbf{d}}+l_y\hat{\mathbf{s}}+l_z\hat{\mathbf{p}}, \tag{5}$$
$$\hat{\mathbf{m}} = m_x\hat{\mathbf{d}}+m_y\hat{\mathbf{s}}+m_z\hat{\mathbf{p}}, \tag{6}$$
$$\hat{\mathbf{n}} = n_x\hat{\mathbf{d}}+n_y\hat{\mathbf{s}}+n_z\hat{\mathbf{p}}. \tag{7}$$

Substitution into the equations of the ellipsoids, and noting that in the plane $\hat{\mathbf{r}}\cdot\hat{\mathbf{p}}=0$, gives, for the intersection of each ellipsoid with the plane

$$\mathbf{r}\cdot\left(\left(\frac{l_x^2}{a^2}+\frac{m_x^2}{b^2}+\frac{n_x^2}{c^2}\right)\hat{\mathbf{d}}\hat{\mathbf{d}}+\left(\frac{l_yl_x}{a^2}+\frac{m_ym_x}{b^2}+\frac{n_xn_y}{c^2}\right)\hat{\mathbf{s}}\hat{\mathbf{d}}+\left(\frac{l_xl_y}{a^2}+\frac{m_xm_y}{b^2}+\frac{n_yn_x}{c^2}\right)\hat{\mathbf{d}}\hat{\mathbf{s}}+\left(\frac{l_y^2}{a^2}+\frac{m_y^2}{b^2}+\frac{n_y^2}{c^2}\right)\hat{\mathbf{s}}\hat{\mathbf{s}}\right)\cdot\mathbf{r}=1,$$

http://www.e-lc.org/docs/2008_10_12_23_11_56

which can be written as

$$\mathbf{r}\mathbb{A}\mathbf{r} = 1, \tag{8}$$

where

$$\mathbb{A} = \alpha\hat{\mathbf{d}}\hat{\mathbf{d}} + \beta\hat{\mathbf{s}}\hat{\mathbf{d}} + \beta\hat{\mathbf{d}}\hat{\mathbf{s}} + \gamma\hat{\mathbf{s}}\hat{\mathbf{s}}, \tag{9}$$

and we note that $\mathbb{A}$ is in $2D$, in the space formed by the orthogonal vectors $\hat{\mathbf{d}}$ and $\hat{\mathbf{s}}$.

We next write

$$\mathbb{A} = u\mathbf{I} - v\hat{\mathbf{k}}\hat{\mathbf{k}}. \tag{10}$$

If u, v and $\hat{\mathbf{k}}$ are determined, the equation of the ellipse in the plane is obtained in standard form [4, 5]. We write

$$\hat{\mathbf{k}} = \cos\phi\hat{\mathbf{d}} + \sin\phi\hat{\mathbf{s}}.$$

and then

$$\mathbb{A} = (\alpha\hat{\mathbf{d}}\hat{\mathbf{d}} + \beta\hat{\mathbf{s}}\hat{\mathbf{d}} + \beta\hat{\mathbf{d}}\hat{\mathbf{s}} + \gamma\hat{\mathbf{s}}\hat{\mathbf{s}}) = u\mathbf{I} - v(\cos\phi\hat{\mathbf{d}} + \sin\phi\hat{\mathbf{s}})(\cos\phi\hat{\mathbf{d}} + \sin\phi\hat{\mathbf{s}})$$

Solving for ϕ and v and u gives,

$$v = \sqrt{4\beta^2 + (\alpha - \gamma)^2}, \tag{11}$$

$$\phi = \frac{1}{2}\tan^{-1}\left(\frac{-2\beta}{\gamma - \alpha}\right), \tag{12}$$

and

$$u = \gamma + v\sin^2\phi. \tag{13}$$

This enables writing the equations for the ellipses in standard form, and the analytic results of Refs. [4, 5] can be used to determine the distance of closest approach $d(\theta)$ for the two ellipses as function of the orientation θ of the plane. The maximum distance of closest approach of the ellipses is the distance of closest approach d_c of the two ellipsoids.

2.2 Step 3. Maximizing the distance of closest approach of the ellipses as function of orientation of the plane

2.2.1 Uniqueness of the maximum

As the plane is rotated about the line joining the centers of the ellipsoids, the distance of closest approach of the ellipses has only one maximum and one minimum as function of the angle of rotation in the interval $[0, \pi)$.

Consider the intersection of the two ellipsoids with the plane with arbitrary orientation. The distance of closest approach of the ellipses is $d'(< d_c)$. If the two ellipsoids are now placed so that their centers are separated by the distance d', they will interpenetrate, and their intersection will be one or two simple closed curves. Since the two ellipses on the plane are now tangent to each other at the point of contact, this point must be on one of the intersection curve as well as on the plane. If there are two intersection curves, this point will be on the curve which is closer to the line of centers. Furthermore, this point is the only point that the plane shares with the intersection curve. That is because if the plane were to share two points with the intersection curve, then there should be either another two ellipses on the plane tangent to each other, which is geometrically impossible; or the two ellipses share two common points, which contradicts the fact that they are tangent. Thus the plane has only one point in common with intersection curve. Since the plane contains only one point on the intersection curve, there are at most two orientations θ of plane containing only one point on the intersection curve.

Therefore there are at most two values of θ giving the same distance d', which guarantees that there is a unique maximum and a unique minimum within the interval $[0, \pi)$.

2.2.2 A fast algorithm to locate the maximum

Standard numerical methods exist to find the extrema of functions. For example, the line search algorithm is an efficient method of unconstrained optimization [6]. More efficient special schemes exist which exploit special properties of functions. The golden section search is a very fast scheme to determine the maximum of a unimodal function which possesses only a single extremum - the maximum [7]. Since $d(\theta)$ possesses a maximum as well as a minimum, the golden section search is not applicable. We have developed a fast algorithm, a modified version of the golden section search, to find the maximum of a function with a single maximum and a single minimum.

We note that the maximum value of must occur between $\theta_1 = 0$ and $\theta_2 = \pi$. As in the golden section search, we choose two points θ_3 and θ_4 inside the interval $[\theta_1, \theta_2)$, and they must satisfy

$$\theta_4 - \theta_1 = \theta_2 - \theta_3 = \frac{1}{\varphi}(\theta_2 - \theta_1), \quad \varphi = \frac{1+\sqrt{5}}{2}. \tag{14}$$

We next examine the relative positions of the first three leftmost values of the function. The three possible scenarios are illustrated in Fig. 2. In two of them, Figs. 2a, 2b, the interval size can be reduced by discarding one subinterval at one end. In the next step, the remaining three points are re-labeled, and one more point is chosen from the larger subinterval according to Eq. (14). In the third case, Fig. 2c, the derivative at the first point must be evaluated in order to decide in which of the two subintervals the maximum is located. If the derivative is positive, the maximum is in the subinterval immediately to the right of the first point. If it is negative, the maximum is in the second, disjoint subinterval. If the derivative is zero, the maximum is at the first point. Once the subinterval containing the maximum is identified, the appropriate subintervals are discarded, and two new points are chosen according to Eq. (14) in the remaining subinterval. The process is then repeated until the size of interval reaches specified tolerance. This algorithm is fast, with the same speed of convergence as the golden section search, i.e., the size of the search interval converges linearly to zero.

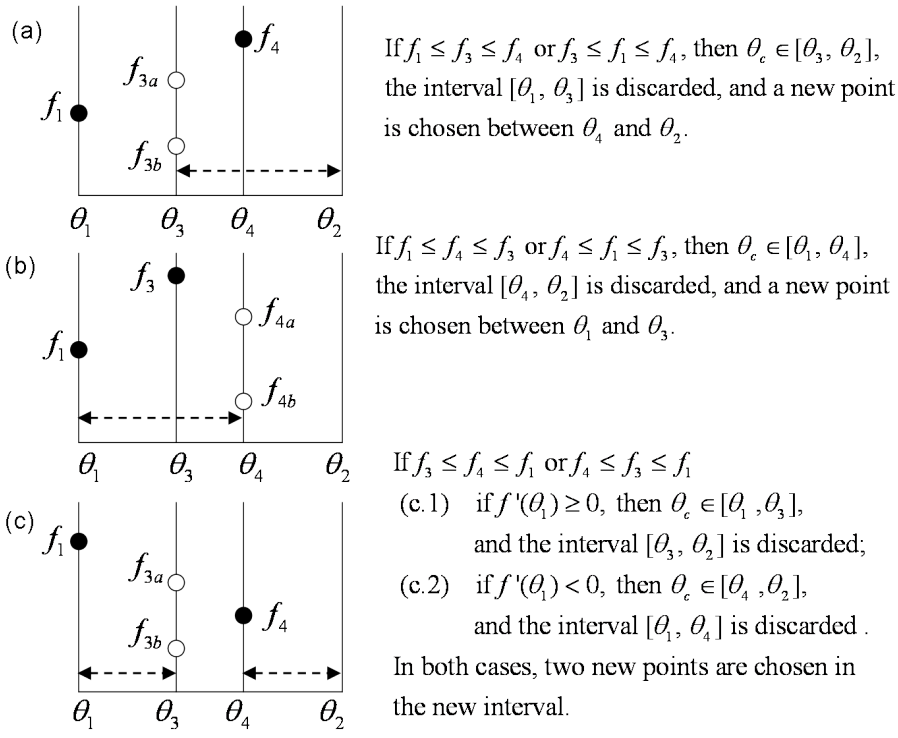

Figure 2: Diagram illustrating scenarios for identifying the subinterval containing the maximum.

http://www.e-lc.org/docs/2008_10_12_23_11_56

2.3 Computational details

To make the results presented in this paper more accessible, we have provided our source code in Fortran and C implementing this algorithm in the Appendix. In both programs, we used double precision throughout.

We note that in our implementations there may be a loss of accuracy for ellipsoids with large aspect ratios (e.g, ≥ 200) when using double precision. In the ellipse program, when the aspect ratio gets large, the ratios of the coefficients in the quartic equation get extremely large, and large number cancelations and/or rounding errors can lead to inaccurate results. If the aspect ratios of the ellipsoids are large (≥ 200), quadruple precision should be used. Benchmarks of computation time are given in the Appendix.

3 Conclusion

We have developed a new algorithm to calculate the distance of closest approach of two arbitrary hard ellipsoids. The algorithm is based on analytic results in the 2D case; it consists of determining the distance of closest approach of two ellipses formed by the intersection of a plane with the ellipsoids as the plane is rotated. The distance of closest approach of the ellipsoids is the maximum distance of closest approach of the ellipses. We have shown that there is only a single maximum, and have developed a fast algorithm to find it. We expect these results to be useful in theoretical and numerical studies of condensed matter systems consisting of ellipsoidal particles.

Acknowledgment We acknowledge effort sponsored by the the National Science Foundation through grants DMR-0606357, and DMS-0821071.

References

[1] W.M. Gelbart and A. Gelbart, "Effective one-body potentials for orientationally anisotropic fluids", Mol. Phys., **33**, 1387-1398 (1977).

[2] J. Vielliard-Baron, "Phase transition of the classical hard ellipse system", *J. Chem. Phys.*, **56**(10), 4729-4744 (1972).

[3] J. W. Perram, M.S. Wertheim, "Statistical mechanics of hard ellipsoids. I. overlap algorithm and the contact function", *J. Comput. Phys.*, **58**, 409-416 (1985).

[4] Zheng and P. Palffy-Muhoray. "Distance of closest approach of two arbitrary hard ellipses in two dimensions", *Phys. Rev. E*, **75**,061709 (2007).

[5] X. Zheng and P. Palffy-Muhoray. "Distance of closest approach of two arbitrary hard ellipses in two dimensions", http://www.e-lc.org/docs/2007_01_17_00_46_52, (2007).

[6] J. Nocedal and S.J. Wright, **Numerical Optimization**, Springer, (2006).

[7] J. Kiefer,"Sequential minimax search for a maximum", *Proc. Am. Math. Soc.*, 4: 502–506 (1953).

A Fortran subroutines

The following Fortran90 subroutine "ellipsoids" requires the following double precision input parameters listed in the constant module:

- a1, b1, c1, a2, b2, c2 - lengths of semiaxes of the two ellipsoids;
- l1(3), m1(3), n1(3), l2(3), m2(3), n2(3) - vectors along the three principal axes of the two ellipsoids;
- d(3) - the vector along the line connecting the centers of the two ellipsoids.

User provided vectors need not to be unit vectors, the subroutine normalizes them.
The output is:

- dist - a double precision variable corresponding to the distance of closest approach of the two ellipsoids.

The subroutine "ellipsoids" includes the fast algorithm for finding the maximum value of the distance between ellipses as function of the angle of rotation of the intersecting plane. It calls another subroutine "plane_ ellp", which takes the input parameter theta which specifies the orientation of the plane, and gives the distance of closest approach of the two ellipses corresponding to the intersection of the plane and the ellipsoids. The subroutine "ellipses" is the program which evaluates the analytic result for the distance of closest approach of two ellipses. A single call to "ellipsoids" with Intel Fortran compiler is typically completed in 0.09 ms on a Red Hat Linux v5 64-bit with Intel Xeon 3.2GHz processor, 2GB RAM.

```
!*****************************************
!Common Variables
!*****************************************
MODULE Common
Implicit None
Double Precision::a1, b1, c1, a2, b2 ,c2 ! length of semiaxes
Double Precision::l1(3), m1(3), n1(3), l2(3), m2(3), n2(3), d(3), p0(3), p0d(3) ! all vectors
END MODULE Common
!*****************************************
!Subroutine to calculate the distance of closest approach of two ellipsoids
!*************************************
Subroutine ellipsoids(dist)
Use Common
Implicit None
Double Precision,Parameter::tau=1.61803398
Double Precision::theta1,theta2,theta3,theta4,pi
Double Precision::dist,f1,f2,f3,f4,f5,f6,err,h=1.D-05
Double Precision,External::norm
pi=atan(1.d0)*4.d0
!normalize the vectors
l1=l1/norm(l1)
m1=m1/norm(m1)
n1=n1/norm(n1)
l2=l2/norm(l2)
m2=m2/norm(m2)
n2=n2/norm(n2)
d=d/norm(d)
!get p0
call cross(d,l1,p0)
If(norm(p0)<1.0d-14) then
     call cross(d,m1,p0)
End If
p0=p0/norm(p0)
```

```
call cross(p0,d,p0d)
!initialize my choice of theta
theta1=0.d0
theta2=pi
theta3=theta2-1.d0/tau*(theta2-theta1)
theta4=theta1+1.d0/tau*(theta2-theta1)
call plane_ellp(theta1,f1)
call plane_ellp(theta3,f3)
call plane_ellp(theta4,f4)
Do while(abs(theta2-theta1)>1.D-9)
    !******************case a
    If((f1<=f3 .and. f3<=f4) .or. (f3<=f1 .and. f1<=f4)) then
        theta1=theta3
        f1=f3
        theta3=theta4
        f3=f4
        theta4=theta1+1.d0/tau*(theta2-theta1)
        call plane_ellp(theta4,f4)
    !******************case b
    ElseIf((f1<=f4 .and. f4<=f3) .or. (f4<=f1 .and. f1<=f3)) then
        theta2=theta4
        theta4=theta3
        f4=f3
        theta3=theta2-1.d0/tau*(theta2-theta1)
        call plane_ellp(theta3,f3)
    !******************case c
    ElseIf((f3<=f4 .and. f4<=f1) .or. (f4<=f3 .and. f3<=f1)) then
        call plane_ellp(theta1+h,f5)
        If(f5>f1) then
            theta2=theta3
            theta3=theta2-1.d0/tau*(theta2-theta1)
            theta4=theta1+1.d0/tau*(theta2-theta1)
            call plane_ellp(theta3,f3)
            call plane_ellp(theta4,f4)
        Else
            call plane_ellp(theta1-h,f6)
            if(f6<f1) then
                theta2=theta3
                theta3=theta2-1.d0/tau*(theta2-theta1)
                theta4=theta1+1.d0/tau*(theta2-theta1)
                call plane_ellp(theta3,f3)
                call plane_ellp(theta4,f4)
            else
                theta1=theta4
                f1=f4
                theta3=theta2-1.d0/tau*(theta2-theta1)
                theta4=theta1+1.d0/tau*(theta2-theta1)
                call plane_ellp(theta3,f3)
                call plane_ellp(theta4,f4)
            end if
        End if
    End If
End Do
call plane_ellp(theta1,dist)
```

```
End Subroutine ellipsoids
!*********************************************************************
!Subroutine to identify the intersection between plane and ellipsoids and
!and return the distance of closest approach of two ellipses
!*********************************************************************
Subroutine plane_ellp(theta,dist)
Use Common
Implicit None
Double Precision::theta
Double Precision::k1(2),k2(2)
Double Precision::l1x,l1y,m1x,m1y,n1x,n1y,alpha1,beta1,gamma1,v1,u1,phi1
Double Precision::l2x,l2y,m2x,m2y,n2x,n2y,alpha2,beta2,gamma2,v2,u2,phi2
Double Precision::ae1,be1,ae2,be2, k1k2,k1d,k2d,dist !arguments needed for closet approach of 2D ellipses
Double Precision::p(3),s(3) !vectors p and s
!get normal to the plane
p=dcos(theta)*p0+dsin(theta)*p0d
call cross(p,d,s)
!generating the first ellipse
call dot(l1,d,l1x)
call dot(l1,s,l1y)
call dot(m1,d,m1x)
call dot(m1,s,m1y)
call dot(n1,d,n1x)
call dot(n1,s,n1y)
alpha1=l1x**2/a1**2+m1x**2/b1**2+n1x**2/c1**2
beta1=l1y*l1x/a1**2+m1y*m1x/b1**2+n1x*n1y/c1**2
gamma1=l1y**2/a1**2+m1y**2/b1**2+n1y**2/c1**2
v1=dsqrt(4.d0*beta1**2+(alpha1-gamma1)**2)
phi1=0.5d0*datan2(-2*beta1,-(alpha1-gamma1))
u1=gamma1+v1*(dsin(phi1))**2
k1(1)=dcos(phi1)
k1(2)=dsin(phi1)
be1=1.d0/dsqrt(u1)
ae1=1.d0/dsqrt(u1-v1)
!generating the second ellipse
call dot(l2,d,l2x)
call dot(l2,s,l2y)
call dot(m2,d,m2x)
call dot(m2,s,m2y)
call dot(n2,d,n2x)
call dot(n2,s,n2y)
alpha2=l2x**2/a2**2+m2x**2/b2**2+n2x**2/c2**2
beta2=l2y*l2x/a2**2+m2y*m2x/b2**2+n2x*n2y/c2**2
gamma2=l2y**2/a2**2+m2y**2/b2**2+n2y**2/c2**2
v2=dsqrt(4.d0*beta2**2+(alpha2-gamma2)**2)
phi2=0.5d0*datan2(-2.*beta2,-(alpha2-gamma2))
u2=gamma2+v2*(dsin(phi2))**2
k2(1)=dcos(phi2)
k2(2)=dsin(phi2)
be2=1.d0/dsqrt(u2)
ae2=1.d0/dsqrt(u2-v2)
!Finding the dot product between the vectors
k1k2=k1(1)*k2(1)+k1(2)*k2(2)
k1d=k1(1)
```

```
k2d=k2(1)
! call 2D subroutine to get the closest approach of ellipses
call ellipses(ae1,be1,ae2,be2,k1k2,k1d,k2d,dist)
End Subroutine plane_ellp
!************************************
!Subroutine to evaluate the dot product of two vectors
!************************************
Subroutine dot(v,u,vdu)
Implicit None
Double Precision::v(3),u(3),vdu
vdu=v(1)*u(1)+v(2)*u(2)+v(3)*u(3)
End Subroutine dot
!************************************
!Subroutine to evaluate the cross product of two vectors
!************************************
Subroutine cross(v,u,vcu)
Implicit None
Double Precision::v(3),u(3),vcu(3)
vcu(1)=v(2)*u(3)-u(2)*v(3)
vcu(2)=u(1)*v(3)-v(1)*u(3)
vcu(3)=v(1)*u(2)-u(1)*v(2)
End Subroutine cross
!************************************
!Function to calculate the length of a vector
!************************************
Double Precision function norm(v)
Implicit None
Double Precision::v(3)
norm=dsqrt(v(1)**2+v(2)**2+v(3)**2)
End function norm
!************************************
!Subroutine to calculate the distance of closest approach of two ellipses
!************************************
Subroutine ellipses(a1,b1,a2,b2,k1k2,k1d,k2d,dist)
Implicit None
Double Complex,Parameter::in=(1.d0,0.d0)
!k1k2, k1d,k2d are dot products between the vectors, k1, k2 and d
Double Precision,Intent(in)::a1,b1,a2,b2,k1k2,k1d,k2d
Double Precision,Intent(out)::dist
Double Precision::e1,e2,eta,a11,a12,a22
Double Precision::lambda1,lambda2,a2p,b2p
Double Precision::kpmp,t,deltap,A,B,C,D,E,alpha,beta,gamma,P,Q,Rp
Double Complex::U,y,qq,z,mysqrt
Double Complex, External::ccbrt
!eccentricity of ellipses
e1=1.d0-b1**2/a1**2
e2=1.d0-b2**2/a2**2
!component of A'
eta=a1/b1-1.d0
a11=b1**2/b2**2*(1.d0+0.5d0*(1.d0+k1k2)*(eta*(2.d0+eta)-e2*(1.d0+eta*k1k2)**2))
a12=b1**2/b2**2*0.5d0*dsqrt(1.d0-k1k2**2)*(eta*(2.d0+eta)+e2*(1.d0-eta**2*k1k2**2))
a22=b1**2/b2**2*(1.d0+0.5d0*(1.d0-k1k2)*(eta*(2.d0+eta)-e2*(1.d0-eta*k1k2)**2))
!eigenvalues of A'
lambda1=0.5d0*(a11+a22)+0.5d0*dsqrt((a11-a22)**2+4.d0*a12**2)
```

```
lambda2=0.5d0*(a11+a22)-0.5d0*dsqrt((a11-a22)**2+4.d0*a12**2)
!major and minor axes of transformed ellipse
b2p=1.d0/dsqrt(lambda1)
a2p=1.d0/dsqrt(lambda2)
deltap=a2p**2/b2p**2-1.d0
If(abs(k1k2)==1.d0) then
    if(a11>a22)then
        kpmp=1.d0/dsqrt(1.d0-e1*k1d**2)*b1/a1*k1d
    elseif(a11<a22) then
        kpmp=dsqrt(1.d0-k1d**2)/dsqrt(1.d0-e1*k1d**2)
    end if
Else
    kpmp=(a12/dsqrt(1.d0+k1k2)*(b1/a1*k1d+k2d+(b1/a1-1.d0)*k1d*k1k2)+ &
    (lambda1-a11)/dsqrt(1.d0-k1k2)*(b1/a1*k1d-k2d-(b1/a1-1.d0)*k1d*k1k2)) &
    /dsqrt(2.d0*(a12**2+(lambda1-a11)**2)*(1.d0-e1*k1d**2))
End If
IF(kpmp==0.d0 .or. deltap==0.0d0) Then
    Rp=a2p+1.d0
ELSE
    !coefficients of quartic for q
    t=1.d0/kpmp**2-1.d0
    A=-1.d0/b2p**2*(1.d0+t)
    B=-2.d0/b2p*(1.d0+t+deltap)
    C=-t-(1.d0+deltap)**2+1.d0/b2p**2*(1.d0+t+deltap*t)
    D=2.d0/b2p*(1.d0+t)*(1.d0+deltap)
    E=(1.d0+t+deltap)*(1.d0+deltap)
    !solution for quartic
    alpha=-3.d0/8.d0*(B/A)**2+C/A
    beta=(B/A)**3.d0/8.d0-(B/A)*(C/A)/2.d0+D/A
    gamma=-3.d0/256.d0*(B/A)**4+C/A*(B/A)**2/16.d0-(B/A)*(D/A)/4.+E/A
    If(beta==0.) Then
        qq=-B/4.d0/A+cdsqrt((-alpha+cdsqrt(alpha**2-4.d0*gamma*in))/2.)
    Else
        P=-alpha**2/12.d0-gamma
        Q=-alpha**3/108.d0+gamma*alpha/3.d0-beta**2/8.d0
        U=ccbrt(-0.5d0*Q+cdsqrt(Q**2/4.d0+P**3/27.d0*in))
        if(abs(U)/=0.0d0) then
            y=-5.d0/6d0*alpha+U-P/3.d0/U
        else
            y=-5.d0/6.d0*alpha-ccbrt(Q)
        end if
        qq=-B/4.d0/A+0.5d0*(cdsqrt(alpha+2.d0*y)+ &
            cdsqrt(-(3.d0*alpha+2.d0*y+2.d0*beta/cdsqrt(alpha+2.d0*y))))
    End If
    Rp=cdsqrt((qq**2-1.d0)/deltap*(1.d0+b2p*(1.d0+deltap)/qq)**2+ &
        (1.d0-(qq**2-1.d0)/deltap)*(1.d0+b2p/qq)**2)
END IF
dist=Rp*b1/dsqrt(1.d0-e1*k1d**2)
End Subroutine ellipses
!*************************************
!Function to evaluate the principal cubic root of a complex number
!*************************************
Double Complex function ccbrt(z)
Implicit None
```

```
Double Complex:: z
Double Precision::arg,theta,r
Double Precision, Intrinsic::datan2
arg=datan2(imag(z),real(z))
theta = arg / 3.0d0
r = (real(z)**2+imag(z)**2)**(1.d0/6.d0)
ccbrt = cmplx(r*dcos(theta), r*dsin(theta))
End function ccbrt
```

B C subroutines

The following C subroutine "ellipsoids" requires the following double precision input parameters declared as global variables:

- a1, b1, c1, a2, b2, c2 - lengths of semiaxes of the two ellipsoids,
- l1i[3], m1i[3], n1i[3], l2i[3], m2i[3], n2i[3] - vectors along the three principal axes of the two ellipsoids,
- di[3] - the vector along the line connecting the centers of the two ellipsoids.

User provided vectors need not to be unit vectors, the subroutine normalizes them.

- sol - a double precision variable corresponding to the distance of closest approach of the two ellipsoids.

This subroutine "ellipsoids" includes the fast algorithm for finding the maximum value of the distance between ellipses as function of the angle of rotation of the intersecting plane. It calls another subroutine"plane_int", which takes the input parameter theta which specifies the orientation of the plane, and gives the distance of closest approach of the two ellipses corresponding to the intersection of the plane and the ellipsoids. The subroutine "distance2D" is the program which evaluates the analytic result for the distance of closest approach of two ellipses. A single call to "ellipsoids" with Intel C compiler is typically completed in 0.09 ms on a Red Hat Linux v5 64-bit with Intel Xeon 3.2GHz processor, 2GB RAM.

```
////////////////////////////////////////////////////////////////////////
////////////////////////////*ellipsoid.h*/////////////////////////////////
////////////////////////////////////////////////////////////////////////
#ifndef ELLIPSOIDS_H_
#define ELLIPSOIDS_H_
#include <stdio.h>
#include<math.h>
#include<errno.h>
#include<stdlib.h>
#include<complex.h>
#define pi 4*atan(1.0)
#define gratio (1.0+sqrt(5.0))*0.5
#define o1g 1.0/(1.0+gratio)
#define tolerance 1E-9
#define delt 1E-5
void crossP(double *x,double *y,double *z); //
void norm(double *vec,double *nvec); //
double mag(double *V); //
double dotP(double *Vec1,double *Vec2); //Functions
double plane_int(double); //
double distance2d(double,double,double,double,double,double); //
double complex c_cbrt(double complex); //
double ellipsoids(void);
double l1i[3],l2i[3],m1i[3],m2i[3],n1i[3],n2i[3],di[3]; //Input vectors
```

```
double a1,a2,b1,b2,c1,c2; //Input semiaxes lenghts
double d[3],p0[3],p[3],s[3],l1[3],l2[3],m1[3],m2[3],n1[3],n2[3],dxp0[3]; //Normalized vectors
#endif
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/*Subroutine to calculate the distance of closest approach of two arbitrary ellipsoids*/
#include "ellipsoids.h"
double ellipsoids(void)
{
    int j;
    double sol;
    double a,b,theta1,theta2,theta3,theta4,f1,f4,f3,f5,f6; //Search algorithm variables
    norm(di,d);
    norm(l1i,l1);
    norm(l2i,l2);
    norm(m1i,m1);
    norm(m2i,m2);
    norm(n1i,n1);
    norm(n2i,n2);
    crossP(d,l1,dxp0);
    if(mag(dxp0)<1.E-14)
        crossP(d,m1,dxp0);
    norm(dxp0,p0);
    crossP(p0,d,dxp0);
    theta1=0.0;
    theta2=pi;
    theta3=theta1+o1g*(theta2-theta1);
    theta4=theta2-o1g*(theta2-theta1);
    f1=plane_int(theta1);
    f3=plane_int(theta3);
    f4=plane_int(theta4);
////////////////////////////search algorithm loop////////////////////////////////
    while(theta2-theta1>tolerance)
    {
        if((f1<=f3 && f3<=f4) || (f3<=f1 && f1<=f4)) //case a
        {
            theta1=theta3;
            f1=f3;
            theta3=theta4;
            f3=f4;
            theta4=theta2-o1g*(theta2-theta1);
            f4=plane_int(theta4);
        }
        else if((f1<=f4 && f4<=f3) || (f4<=f1 && f1<=f3)) //case b
        {
            theta2=theta4;
            theta4=theta3;
            f4=f3;
            theta3=theta1+o1g*(theta2-theta1);
            f3=plane_int(theta3);
        }
        else //case c
        {
            f5=plane_int(theta1+delt);
```

```
                if(f5>f1)
                {
                    theta2=theta3;
                    theta3=theta1+o1g*(theta2-theta1);
                    theta4=theta2-o1g*(theta2-theta1);
                    f3=plane_int(theta3);
                    f4=plane_int(theta4);
                }
                else
                {
                    f6=plane_int(theta1-delt);
                    if(f6<f1)
                    {
                        theta2=theta3;
                        theta3=theta1+o1g*(theta2-theta1);
                        theta4=theta2-o1g*(theta2-theta1);
                        f3=plane_int(theta3);
                        f4=plane_int(theta4);
                    }
                    else
                    {
                        theta1=theta4;
                        f1=f4;
                        theta3=theta1+o1g*(theta2-theta1);
                        theta4=theta2-o1g*(theta2-theta1);
                        f3=plane_int(theta3);
                        f4=plane_int(theta4);
                    }
                }
            }
        }
        sol=f4;
        return sol;
}
////////////////////////////////////////////////////////////////////////////
//////////////////////*FUNCTIONS*///////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/*Cross Product of two vectors*/
void crossP(double x[3],double y[3],double z[3])
{
        z[0]=x[1]*y[2]-x[2]*y[1];
        z[1]=x[2]*y[0]-x[0]*y[2];
        z[2]=x[0]*y[1]-x[1]*y[0];
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/*Normalization of a vector*/
void norm(double vec[3],double nvec[3])
{
        double length;
        length=mag(vec);
        if(length==0.0)
        {
            nvec[0]=0.0;nvec[1]=0.0;nvec[2]=0.0;
```

```
        }
        else
        {
        nvec[0]=vec[0]/length;
        nvec[1]=vec[1]/length;
        nvec[2]=vec[2]/length;
        }
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/*Magnitude of a vector*/
double mag(double V[3])
{
        return sqrt(V[0]*V[0]+V[1]*V[1]+V[2]*V[2]);
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/*Dot Product of two vectors*/
double dotP(double Vec1[3],double Vec2[3])
{
        return Vec1[0]*Vec2[0]+Vec1[1]*Vec2[1]+Vec1[2]*Vec2[2];
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/*Ellipses formed with the intersection of the ellipsoid with the plane*/
double plane_int(double theta)
{
        double alpha1,beta1,gamma1,v1,u1,alpha2,beta2,gamma2,v2,u2;
        double e1,e2;
        double l1x,l1y,m1x,m1y,n1x,n1y,l2x,l2y,m2x,m2y,n2x,n2y;
        double a2d1,a2d2,b2d1,b2d2,angle1,angle2; //Ellipses variables
        int j;
        for(j=0;j<3;j++)
                p[j]=cos(theta)*p0[j]+sin(theta)*dxp0[j];
        crossP(p,d,s);
        l1x=dotP(l1,d);
        l1y=dotP(l1,s);
        m1x=dotP(m1,d);
        m1y=dotP(m1,s);
        n1x=dotP(n1,d);
        n1y=dotP(n1,s);
        alpha1=l1x*l1x/(a1*a1)+m1x*m1x/(b1*b1)+n1x*n1x/(c1*c1);
        beta1=l1y*l1x/(a1*a1)+m1y*m1x/(b1*b1)+n1x*n1y/(c1*c1);
        gamma1=l1y*l1y/(a1*a1)+m1y*m1y/(b1*b1)+n1y*n1y/(c1*c1);
        v1=sqrt(4.0*beta1*beta1+(alpha1-gamma1)*(alpha1-gamma1));
        angle1=0.5*atan2(-2.0*beta1,-(alpha1-gamma1));
        u1=gamma1+v1*sin(angle1)*sin(angle1);
        b2d1=1.0/sqrt(u1);
        a2d1=1./sqrt(u1-v1);
        l2x=dotP(l2,d);
        l2y=dotP(l2,s);
        m2x=dotP(m2,d);
        m2y=dotP(m2,s);
        n2x=dotP(n2,d);
```

```
        n2y=dotP(n2,s);
        alpha2=l2x*l2x/(a2*a2)+m2x*m2x/(b2*b2)+n2x*n2x/(c2*c2);
        beta2=l2y*l2x/(a2*a2)+m2y*m2x/(b2*b2)+n2x*n2y/(c2*c2);
        gamma2=l2y*l2y/(a2*a2)+m2y*m2y/(b2*b2)+n2y*n2y/(c2*c2);
        v2=sqrt(4.0*beta2*beta2+(alpha2-gamma2)*(alpha2-gamma2));
        angle2=0.5*atan2(-2.0*beta2,-(alpha2-gamma2));
        u2=gamma2+v2*sin(angle2)*sin(angle2);
        b2d2=1.0/sqrt(u2);
        a2d2=1.0/sqrt(u2-v2);
        return distance2d(a2d1,b2d1,a2d2,b2d2,angle1,angle2);
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/*Distance of Closest Approach of two arbitrary ellipses*/
double distance2d(double a1, double b1, double a2, double b2, double angle1, double angle2)
{
        double eps1,eps2,k1dotd,k2dotd,k1dotk2,nu,Ap[2][2],
                lambdaplus,lambdaminus,bp2,ap2,cosphi,tanphi2,delta,dp;
        double complex A,B,C,D,E,alpha,beta,gamma,P,Q,U,y,qu;
        eps1=sqrt(1.0-(b1*b1)/(a1*a1));
        eps2=sqrt(1.0-(b2*b2)/(a2*a2));
        k1dotd=cos(angle1);
        k2dotd=cos(angle2);
        k1dotk2=cos(angle2-angle1);
        nu=a1/b1-1.0;
        Ap[0][0]=b1*b1/(b2*b2)*(1.0+0.5*(1.0+k1dotk2)*
                        (nu*(2.0+nu)-eps2*eps2*(1.0+nu*k1dotk2)*(1.0+nu*k1dotk2)));
        Ap[1][1]=b1*b1/(b2*b2)*(1.0+0.5*(1.0-k1dotk2)*
                        (nu*(2.0+nu)-eps2*eps2*(1.0-nu*k1dotk2)*(1.0-nu*k1dotk2)));
        Ap[0][1]=b1*b1/(b2*b2)*0.5*sqrt(1.0-k1dotk2*k1dotk2)*
                        (nu*(2.0+nu)+eps2*eps2*(1.0-nu*nu*k1dotk2*k1dotk2));
        lambdaplus=0.5*(Ap[0][0]+Ap[1][1])+sqrt(0.25*(Ap[0][0]-Ap[1][1])*
                        (Ap[0][0]-Ap[1][1])+Ap[0][1]*Ap[0][1]);
        lambdaminus=0.5*(Ap[0][0]+Ap[1][1])-sqrt(0.25*(Ap[0][0]-Ap[1][1])*
                        (Ap[0][0]-Ap[1][1])+Ap[0][1]*Ap[0][1]);
        bp2=1.0/sqrt(lambdaplus);
        ap2=1.0/sqrt(lambdaminus);
        if(k1dotk2==1.0)
        {
                if(Ap[0][0]>Ap[1][1])
                        cosphi=b1/a1*k1dotd/sqrt(1.0-eps1*eps1*k1dotd*k1dotd);
                else
                        cosphi=sqrt(1.0-k1dotd*k1dotd)/sqrt(1.0-eps1*eps1*k1dotd*k1dotd);
        }
        else
        {
                cosphi=1.0/sqrt(2.0*(Ap[0][1]*Ap[0][1]+(lambdaplus-Ap[0][0])*(lambdaplus-Ap[0][0]))*
                        (1.0-eps1*eps1*k1dotd*k1dotd))*(Ap[0][1]/sqrt(1.0+k1dotk2)*(b1/a1*k1dotd+
                        k2dotd+(b1/a1-1.0)*k1dotd*k1dotk2)+ (lambdaplus-Ap[0][0])/sqrt(1.0-k1dotk2)*
                        (b1/a1*k1dotd-k2dotd-(b1/a1-1.0)*k1dotd*k1dotk2));
        }
        delta=ap2*ap2/(bp2*bp2)-1.0;
        if(delta==0.0 || cosphi==0.0)
                dp=1.0+ap2;
```

```
        else
        {
            tanphi2=1.0/(cosphi*cosphi)-1.0;
            A=-(1.0+tanphi2)/(bp2*bp2);
            B=-2.0*(1.0+tanphi2+delta)/bp2;
            C=-tanphi2-(1.0+delta)*(1.0+delta)+(1.0+(1.0+delta)*tanphi2)/(bp2*bp2);
            D=2.0*(1.0+tanphi2)*(1.0+delta)/bp2;
            E=(1.0+tanphi2+delta)*(1.0+delta);
            alpha=-3.0*B*B/(8.0*A*A)+C/A;
            beta=B*B*B/(8.0*A*A*A)-B*C/(2.0*A*A)+D/A;
            gamma=-3.0*B*B*B*B/(256.0*A*A*A*A)+C*B*B/(16.0*A*A*A)-B*D/(4.0*A*A)+E/A;
            if(beta==0.0)
            {
                qu=-B/(4.0*A)+csqrt(0.5*(-alpha+csqrt(alpha*alpha-4.0*gamma)));
            }
            else
            {
                P=-alpha*alpha/12.0-gamma;
                Q=-alpha*alpha*alpha/108.0+alpha*gamma/3.0-beta*beta/8.0;
                U=c_cbrt(-Q*0.5+csqrt(Q*Q*0.25+P*P*P/27.0));
                if(U==0.0)
                    y=-5.0*alpha/6.0-c_cbrt(Q);
                else
                    y=-5.0*alpha/6.0+U-P/(3.0*U);
                qu=-B/(4.0*A)+0.5*(csqrt(alpha+2.0*y)+
                    csqrt(-(3.0*alpha+2.0*y+2.0*beta/csqrt(alpha+2.0*y))));
            }
            dp=csqrt((qu*qu-1.0)/delta*(1.0+bp2*(1.0+delta)/qu)*
                (1.0+bp2*(1.0+delta)/qu)+(1.0-(qu*qu-1.0)/delta)* (1.0+bp2/qu)*(1.0+bp2/qu));
        }
        return dp*b1/sqrt(1.0-eps1*eps1*k1dotd*k1dotd);
}
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
/* Principal cubic root of a complex number */
double complex c_cbrt(double complex x)
{
        double a,b,r,phi,rn;
        a=creal(x);
        b=cimag(x);
        r=sqrt(a*a+b*b);
        phi=atan2(b,a);
        phi/=3.0;
        rn=cbrt(r);
        return rn*cos(phi)+I*rn*sin(phi);
}
```