Electrostatic polyhedron

Domenico Oricchio

August 23, 2023

Abstract

I minimize the N charges electric potential on a sphere, the minimum potential optimize the distance between the charges and it is possible to obtain the polyhedrons from the N charge positions.

In chemistry happen that some geometric configuration are obtained in molecular structure: tetrahedron for carbon atoms, cubic body centered, hexagonal; the same minimum configuration happen for minimal surface problem solution with soap film; So that some geometrical problems can be solved with an energy minimization.

I set myself the problem of optimizing the direction of the lasers for the confinement of the plasma by searching for the optimal directions of the beams, in order to maximize the angular coverage of the beams

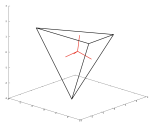
N charges on a sphere can repel each other with a force that lead to the equilibrium with N points which are equidistant from their nearest neighbours.

I use a repulsive potential that is the error function for the minimization problem (gradient descent, conjugate gradient method or other numerical solution):

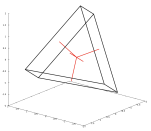
$$U(\boldsymbol{\theta},\boldsymbol{\phi}) = \sum_{1\leq i\leq j\leq N} \frac{1}{\sqrt{(x_i-x_j)^2+(y_i-y_j)^2+(y_i-y_j)^2}}$$
$$x_i = \sin(\theta_i)\cos(\phi_i);$$
$$y_i = \sin(\theta_i)\sin(\phi_i);$$
$$z_i = \cos(\theta_i);$$

an electric charge is not in equilibrium in the vacuum for each electrostatic field[1], but it is possible a equilibrium configuration for charges on a surface (the reason why the electrostatics exist) because of there are other forces, and there is an absolute minimum for the electric potential on the surface.

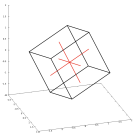
The Lagrance theorem[2] establishes that the absolute minimum of the potential is a stable equilibrium of the mechanical system, so that exist a



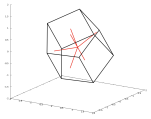
(4) Tetrahedron



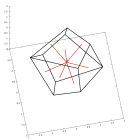
(5) Triangular prism



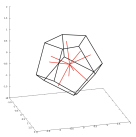
(6) Cube



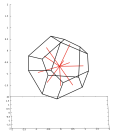
(7) Pentagonal prism



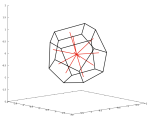
(8) Tetragonal trapezohedron



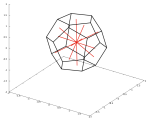
(9) Associahedron t4dP3



(10) Truncated square trapezohedron



(11) Hendecahedra



(12) Regular dodecahedron

configuration where the force applied to the charges are all zero (the gradients that are used in the minimization are the applied forces on the charges in generalized coordinates):

$$F_{\theta_s} = -\frac{\partial U}{\partial \theta_s} = \sum_{j \neq s}^{N} \frac{(x_s - x_j)\cos(\theta_s)\cos(\phi_s) + (y_s - y_j)\cos(\theta_s)\sin(\phi_s) - (z_s - z_j)\sin(\theta_s)}{\sqrt{[(x_s - x_j)^2 + (y_s - y_j)^2 + (y_s - y_j)^2]^3}}$$

$$F_{\phi_s} = -\frac{\partial U}{\partial \phi_s} = \sum_{j \neq s}^{N} \frac{-(x_s - x_j)\sin(\theta_s)\sin(\phi_s) + (y_s - y_j)\sin(\theta_s)\cos(\phi_s)}{\sqrt{[(x_s - x_j)^2 + (y_s - y_j)^2 + (y_s - y_j)^2]^3}}$$

sometimes the evolution of the gradient descent is very slow, due to the shape of the potential (sharp canyon for the potentials), but a random direction for the gradient descent can reduce the problem easily

$$\theta_i^{new} = \theta_i - \sigma\, \mathcal{N}\left(\frac{\partial U}{\partial \theta_s}\right)$$

$$\phi_i^{new} = \phi_i - \sigma\, \mathcal{N}\left(\frac{\partial U}{\partial \phi_s}\right)$$

$$\begin{cases} if\ U^{new} < U \quad then \quad U = U^{new}, \boldsymbol{\theta} = \boldsymbol{\theta}^{new}, \boldsymbol{\phi} = \boldsymbol{\phi}^{new}, \sigma = 1.21\sigma^{new} \\ \\ if\ U^{new} \geq U \quad then \quad \sigma = \sigma^{new}/1.1 \end{cases}$$

where $\mathcal{N}$ is a random variable with a normal distribution with mean 0 and variance η.

The polyhedron faces can be calculate obtaining the vertices of the polyhedron using the intersections of the three planes tangent to the sphere and passing through the charges (the charges are identified by a vector pointing from the center of the unitary sphere to the charge): the three -or more- orthogonal planes, tangent to the sphere intersect the vertex (x, y, z):

$$\begin{aligned} 0 &= x_i(x - x_i) + y_i(y - y_i) + z_i(z - z_i) \\ 1 &= x_i x + y_i y + z_i z \end{aligned}$$

indeed the tangent plane passes through the coordinates of the charge, and has zero scalar product with the vector of the charge; the equation for the intersection of the three planes are:

$$\begin{aligned} 1 &= x_i x + y_i y + z_i z \\ 1 &= x_j x + y_j y + z_j z \\ 1 &= x_k x + y_k y + z_k z \end{aligned}$$

and the vertex is:

$$(x, y, z) = \left(\frac{\begin{vmatrix} 1 & y_i & z_i \\ 1 & y_j & z_j \\ 1 & y_k & z_k \end{vmatrix}}{\begin{vmatrix} x_i & y_i & z_i \\ x_j & y_j & z_j \\ x_k & y_k & z_k \end{vmatrix}}, \frac{\begin{vmatrix} x_i & 1 & z_i \\ x_j & 1 & z_j \\ x_k & 1 & z_k \end{vmatrix}}{\begin{vmatrix} x_i & y_i & z_i \\ x_j & y_j & z_j \\ x_k & y_k & z_k \end{vmatrix}}, \frac{\begin{vmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{vmatrix}}{\begin{vmatrix} x_i & y_i & z_i \\ x_j & y_j & z_j \\ x_k & y_k & z_k \end{vmatrix}} \right)$$

it is necessary to have a function to choose the different vector to obtain the vertices; if the charge choice is optimal, then the volume of the polyhedron face identify a section of a sphere with a volume that must be (N is the charge number and R=1):

$$V = \frac{4}{3N}\pi R^3$$

a spherical sector must have the same volume, to estimate the maximum angle between the charge vectors:

$$V = \frac{2\pi R}{3}\left[1 - \cos(\phi)\right]$$

where ϕ is half of the cone angle: if each scalar product of two charge vectors in the triad of vector is greater of $\cos(\phi)$ then the triad give a vertex of the polyhedron: it is possible calculated by hand with gnuplot the vertex

position using different color for colored vertex vectors.

The polyhedron evaluation is important because of it is possible to compare classical geometric results, and to evaluate coincidences and variations: choosing uniform polyhedron to identify directions may not be the best choice

There are particular results for icoesahedron (different from what one would expect), and it is possible to choose a solution where each charge (x_i, y_i, z_i) has a charge $(-x_i, -y_i, -z_i)$ in the opposite positions, so that the laser beam has perfect symmetry.

Language C program

```
/** ************************************************************ **
*             laser_10.c - ELECTROSTATIC POLYHEDRON              *
*                    Copyright (C) 2014                          *
*    Oricchio Domenico - Theorical Physicist from Salerno University  *
*                                                                *
* This program is free software; you can redistribute it and/or  *
* modify it under the terms of the GNU General Public License    *
* as published by the Free Software Foundation; either version 2 *
* of the License, or (at your option) any later version.         *
*                                                                *
* This program is distributed in the hope that it will be useful, *
* but WITHOUT ANY WARRANTY; without even the implied warranty of  *
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the   *
* GNU General Public License for more details.                   *
*                                                                *
* You should have received a copy of the GNU General Public      *
* License along with this program; if not, write to the Free     *
* Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, *
* MA 02111-1307, USA                                             *
** ************************************************************ **/
#include<assert.h>
#include <ctype.h>
#include <errno.h>
#include <float.h>
#include<limits.h>
#include<locale.h>
#include  <math.h>
#include<setjmp.h>
#include<signal.h>
#include<stdarg.h>
#include<stddef.h>
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
#include  <time.h>
#include<unistd.h>

/** Gaussian random generator Marsiglia polar method */
long double Gammal( long double sigma);

int main(   int     argc,
            char    **argv)
{
    int                 i,
                j,
                k,
                n=10;
    long int    step=0L,
                step_max=1000000;
                long double     E,
                E_bst,
```

```
                r_1,
                eta=1.0L,
                V,
                norm_grad,
                *theta, *theta_bst, *d_theta,
                *phi, *phi_bst, *d_phi,
                *x, *x_bst,
                *y, *y_bst,
                *z, *z_bst;

    for( i=1; i<argc; i++)
    {
        if( strcmp( argv[i], "-n" )==0 ){ n        = atoi( argv[i+1]); }
        if( strcmp( argv[i], "-s" )==0 ){ step_max = atoi( argv[i+1]); }
    }
    printf( "\nn=%d s=%ld\n\n", n, step_max );
    if( (theta     = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE theta[]     \n");
        exit(EXIT_FAILURE); }
    if( (theta_bst = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE theta_bst[]\n");
        exit(EXIT_FAILURE); }
    if( (d_theta   = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE d_theta[]   \n");
        exit(EXIT_FAILURE); }

    if( (phi       = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE phi[]       \n");
        exit(EXIT_FAILURE); }
    if( (phi_bst   = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE phi_bst[]   \n");
        exit(EXIT_FAILURE); }
    if( (d_phi     = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE d_phi[]     \n");
        exit(EXIT_FAILURE); }

    if( (x         = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE x[]         \n");
        exit(EXIT_FAILURE); }
    if( (x_bst     = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE x_bst[]     \n");
        exit(EXIT_FAILURE); }

    if( (y         = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE y[]          \n");
        exit(EXIT_FAILURE); }
    if( (y_bst     = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE y_bst[]     \n");
        exit(EXIT_FAILURE); }

    if( (z         = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE z[]         \n");
        exit(EXIT_FAILURE); }
    if( (z_bst     = (long double *)calloc(n, sizeof(long double)))==NULL){ perror( "I CANNOT ALLOCATE z_bst[]     \n");
        exit(EXIT_FAILURE); }
    do{
        for( i=0, E_bst=norm_grad=0.0L; i<n; i++ )
        {
/** initial face center of the polyhedron in spherical coordinates */
            theta_bst[i] =       (long double)M_PI* (2.0L*(long double)rand()/ (long double)RAND_MAX- 1.0L);
            phi_bst[i]   = 2.0L* (long double)M_PI*       (long double)rand()/ (long double)RAND_MAX;
            x_bst[i]     = sinl(theta_bst[i])* cosl(phi_bst[i]);
            y_bst[i]     = sinl(theta_bst[i])* sinl(phi_bst[i]);
            z_bst[i]     = cosl(theta_bst[i]);
            for( j=0, d_theta[i]=d_phi[i]=0.0L; j<n; j++ )
            {
/** the central point of the polyhedron surface repulse each other with a simple potential */
                if( i!=j )
                {
                    r_1         = 1.0L/ sqrt((x_bst[i]-x_bst[j])* (x_bst[i]-x_bst[j])+ (y_bst[i]-y_bst[j])* (y_bst[i]-
                        y_bst[j])+ (z_bst[i]-z_bst[j])* (z_bst[i]-z_bst[j]));
                    if( j<i ){ E_bst += r_1; }
                    d_theta[i] -= r_1* r_1* r_1* ( (x_bst[i]-x_bst[j])* cosl(theta_bst[i])* cosl(phi_bst[i])
                                    +(y_bst[i]-y_bst[j])* cosl(theta_bst[i])* sinl(phi_bst[i])
                                    -(z_bst[i]-z_bst[j])* sinl(theta_bst[i]) );
                    d_phi[i]   -= r_1* r_1* r_1* (-(x_bst[i]-x_bst[j])* sinl(theta_bst[i])* sinl(phi_bst[i])
                                    +(y_bst[i]-y_bst[j])* sinl(theta_bst[i])* cosl(phi_bst[i]) );
                }
            }
            norm_grad += (d_theta[i]*d_theta[i]+ d_phi[i]*d_phi[i] );
        }
        norm_grad = sqrtl(norm_grad);
    }while( !isfinite(E_bst) || !isfinite(norm_grad) );
    for( step=0L; step<step_max; step++ )
    {
        for( i=0, E=0.0L; i<n; i++ )
        {
```

```
/** random gradient descent: the dynamis is a random choice using the gradient direction */
            theta[i] = theta_bst[i]+ eta* Gammal(d_theta[i]);
            phi[i]   = phi_bst[i]  + eta* Gammal(d_phi[i]  );
            x[i]     = sinl(theta[i])* cosl(phi[i]);
            y[i]     = sinl(theta[i])* sinl(phi[i]);
            z[i]     = cosl(theta[i]);
            for( j=0; j<i; j++ )
            {
                E += 1.0L/sqrt((x[i]-x[j])*(x[i]-x[j])+ (y[i]-y[j])*(y[i]-y[j])+ (z[i]-z[j])*(z[i]-z[j]));
            }
        }
        if( E<=E_bst && isfinite(E) )
        {
            printf( "\rstep=%8ld eta=%Le E=%Le E_bst=%.*Le D=%Le     ", step_max-step, eta, E, LDBL_DIG, E_bst,
                 norm_grad);
            E_bst = E;
            for( i=0; i<n; i++ )
            {
                theta_bst[i] = theta[i];
                phi_bst[i]   = phi[i];
                x_bst[i]     = x[i];
                y_bst[i]     = y[i];
                z_bst[i]     = z[i];
            }
/** if the potential decrease, then the gaussian deviation is increased */
            if( E<E_bst ){ eta *= 1.21L;  }
            else         { eta *= 0.90909L;}
            for( i=0, norm_grad=0.0L; i<n; i++ )
            {
                for( j=0, d_theta[i]=d_phi[i]=0.0L; j<n; j++ )
                {
                    if( i!=j )
                    {
                        r_1          = 1.0L/sqrt((x_bst[i]-x_bst[j])* (x_bst[i]-x_bst[j])+ (y_bst[i]-y_bst[j])* (y_bst[i]-
                             y_bst[j])+ (z_bst[i]-z_bst[j])* (z_bst[i]-z_bst[j]));
                        d_theta[i] -= r_1* r_1* r_1* (   (x_bst[i]-x_bst[j])* cosl(theta_bst[i])* cosl(phi_bst[i])
                                                        +(y_bst[i]-y_bst[j])* cosl(theta_bst[i])* sinl(phi_bst[i])
                                                        -(z_bst[i]-z_bst[j])* sinl(theta_bst[i]) );
                        d_phi[i]   -= r_1* r_1* r_1* (  -(x_bst[i]-x_bst[j])* sinl(theta_bst[i])* sinl(phi_bst[i])
                                                        +(y_bst[i]-y_bst[j])* sinl(theta_bst[i])* cosl(phi_bst[i]) );
                    }
                }
                norm_grad += ( d_theta[i]*d_theta[i]+ d_phi[i]*d_phi[i] );
            }
            norm_grad = sqrtl( norm_grad);
        }
        else
        {
            eta *= 0.90909L;
        }
/** if there is not gaussian evolution (too little deviation), then the program restart the gaussian with great value */
        if( eta<=LDBL_EPSILON ){ eta = 1.0/norm_grad; printf( "\rstep=%8ld eta=%Le E=%Le E_bst=%.*Le D=%Le   \n",
             step_max-step, eta, E, LDBL_DIG, E_bst, norm_grad ); }
    }
    printf("\n");
    for( i=0; i<n; i++)
    {
        printf( "t=%Le p=%Le x=%Le y=%Le z=%Le\n", theta_bst[i], phi_bst[i], x_bst[i], y_bst[i], z_bst[i]);
    }
/** gnuplot can be used to draw the central point of the surface of the polyhedron */
    printf( "\ngnuplot: cut and paste in the gnuplot command line\nreset\nset xrange [%lg:%lg]\nset yrange [%lg:%lg]\
         nset zrange [%lg:%lg]\nset view equal xyz\nset xyplane relative 0\n", -2.0, 2.0, -2.0, 2.0, -2.0, 2.0 );
    for( i=0, step=0L; i<n; i++ )
    {
        step++; printf( "set arrow %3ld from %Lg, %Lg, %Lg  to 0,0,0 nohead lw 3 lc 7\n", step, x_bst[i], y_bst[i],
             z_bst[i]);
    }
    printf("\n");
    eta = 2.0/ sqrtl( (long double)n );
    for( i=0, eta=1.0L-2.0L/sqrtl((long double)n); i<n; i++ )
    {
        for( j=0; j<i; j++ )
        {
            for( k=0, eta=0.0; k<j; k++ )
            {
                if( x_bst[i]*x_bst[j]+ y_bst[i]*y_bst[j] +z_bst[i]*z_bst[j]>=eta &&
```

```
                    x_bst[i]*x_bst[k]+ y_bst[i]*y_bst[k] +z_bst[i]*z_bst[k]>=eta &&
                    x_bst[k]*x_bst[j]+ y_bst[k]*y_bst[j] +z_bst[k]*z_bst[j]>=eta )
                {
                    V    =  x_bst[i]* (y_bst[j]*z_bst[k]- y_bst[k]*z_bst[j])
                           -x_bst[j]* (y_bst[i]*z_bst[k]- y_bst[k]*z_bst[i])
                           +x_bst[k]* (y_bst[i]*z_bst[j]- y_bst[j]*z_bst[i]);
                    x[i] = ( (y_bst[j]*z_bst[k]- y_bst[k]*z_bst[j])
                            -(y_bst[i]*z_bst[k]- y_bst[k]*z_bst[i])
                            +(y_bst[i]*z_bst[j]- y_bst[j]*z_bst[i]) )/V;
                    y[i] = (-(x_bst[j]*z_bst[k]- x_bst[k]*z_bst[j])
                            +(x_bst[i]*z_bst[k]- x_bst[k]*z_bst[i])
                            -(x_bst[i]*z_bst[j]- x_bst[j]*z_bst[i]) )/V;
                    z[i] = ((x_bst[j]*y_bst[k]- x_bst[k]*y_bst[j])
                            -(x_bst[i]*y_bst[k]- x_bst[k]*y_bst[i])
                            +(x_bst[i]*y_bst[j]- x_bst[j]*y_bst[i]) )/V;
                    step++; printf( "set arrow %3ld from % Lg, % Lg, % Lg  to 0,0,0 nohead lw 3 lc 6\n", step, x[i], y[i],
                          z[i]);
                }
            }
        }
    }
    printf( "splot NaN notitle\n" );
    exit( EXIT_SUCCESS);
    return( EXIT_SUCCESS);
}

long double Gammal( long double sigma)
{
    long double u,
                s,
                v;
    do{
        u = 2.0L* (long double)rand()/ (long double)RAND_MAX- 1.0L;
        v = 2.0L* (long double)rand()/ (long double)RAND_MAX- 1.0L;
        s = u* u+ v* v;
    }while( s>=1.0L || s==0.0L );
    s = sqrtl( -2.0L* logl(s)/ s );
    return( sigma* u* s );
}
```

References

[1] Electricity and Magnetism, Edward M. Purcell, 1986, McGraw-Hill Education.

[2] Lectures in Analytical Mechanics, F.R. Gantmacher, 1970, Mir Publisher