**Supplementary Information**

## S-2. Calculation procedures

### Part A. Electrostatic potential energy change with ion transfer between blend domains.

Calculations were done using a code written in Matlab® language. The program is divided in three parts: “partition_controller”, “patition_core” and “partition_energy”. The first asks the user for input data and calls the second program. “Partition_core” builds the virtual object that contains charges, performs charge migration changes and calls “partition_energy”. This last subroutine calculates electrostatic potential energy of the virtual object.

The virtual object is formed by two adjacent cubes, each formed by 125 cells (5×5×5) separated from their neighbors by 1 nm along each Cartesian axis. Each cell may contain a positive or a negative or a null unit charge (1 unit = $1.602\times10^{-19}$ C).

Charge assignment was done according to the following procedure:

1) Sample potential histograms were obtained from the SEPM images of the separate polymers (Figures 1b and 1d).

2) In every case, histograms showed a bimodal potential distribution and each peak was fitted using a first order Gaussian. The distribution for the relevant polymer domains (positive in natural rubber and negative in P(S-BA) yielded the average electric potential for each polymer.

3) Charge ratio in the two polymers was taken as equal to the ratio of electrostatic potential averages. The average electric potential for natural rubber is equal to +1.91 V and for P(S-BA) is –0.07 V. Thus, the ratio of charges in the two cubes of the virtual object was taken as +1.91/–0.07. This means that the cube representing natural rubber contained *ca.* 27 positive charges per negative charge contained in P(S-BA) cube. The fraction of the cells occupied by positive charges in each cube is represented by *g* and the fraction of cells occupied by negative

charges in each cube is *h* and these values are entered as initial inputs in each cube.

The electric potentials considered in 2) are mean values of a Gaussian curves. Therefore, both *h* and *g* were allowed to fluctuate within the curve around the input value, to better simulate charge distribution in the cells of the virtual object. For this reason, each initial configuration has different numbers of charges, yielding different ΔE vs. percentage migration curves shown in Figure S2a and c. The average curve for all 120 different initial configurations (Figure S2b and d) was considered representative of the sample because the average charge thus obtained agrees with experimental data.

Input volume fractions are shown in Table S2 and they are in agreement with the charge ratio calculated from SEPM data.

**Table S2.** Input values of volumetric fraction of charge for charge migration calculations.

| Volume fraction | Natural rubber cube | P(S-BA) cube | Charge ratio |
|---|---|---|---|
| neutral charges (f) | 0.4542 | 0.98 | $\frac{0.5458}{0.02} = 27$ |
| negative charges (h) | 0 | 0.02 | |
| positive charges (g) | 0.5458 | 0 | |

The location of each point-like charge within both cubes was randomly chosen in order to simulate any possible arrangement of polymer chains and ionic species at the interfacial region of real material. The large number of possible combinations gives numerous possibilities of system initial states. For this reason, 120 initial configurations were built to yield the effect of charge migration considering many different initial states.

Charges were then allowed to migrate from each configuration of the NR cube to the P(S-BA) cube, one by one. The electric potential energy was calculated as the sum of point-like energy of the interaction of each charge with all other charges (Equation 1), according to the superposition principle.

$$E_T = \frac{1}{4\pi\varepsilon_0}\left(\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{q_{i(NR)}q_{j(NR)}}{\varepsilon_{NR}r_{ij}} + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{q_{i(PSBA)}q_{j(PSBA)}}{\varepsilon_{PSBA}r_{ij}} + \sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{q_{i(NR)}q_{j(PSBA)}}{\varepsilon_{NR/PSBA}r_{ij}}\right) \quad (1)$$

Equation 1 represents the sum of interactions between charges within NR cube, added to the sum of interactions of charges inside P(S-BA) cube and the interactions between the charges in one cube with charges in the other cube; $r_{ij}$ is the distance between the charges *i* and *j*; $\varepsilon_{NR}$ and $\varepsilon_{PSBA}$ are the dielectric constants of natural rubber and P(S-BA) domains, respectively, and $\varepsilon_{NR/PSBA}$ is the combined dielectric constant used to evaluate the interaction between charges located in the two different media. $\varepsilon_{NR/PSBA}$ is taken as the weighted average of $\varepsilon_{NR}$ and $\varepsilon_{PSBA}$ where the weights are the distances between each charge and the interface (Equation 2).

$$\varepsilon_{NR/PSBA} = \frac{\varepsilon_{NR}\cdot d_{i(NR)\leftrightarrow \mathrm{int}} + \varepsilon_{PSBA}\cdot d_{j(PSBA)\leftrightarrow \mathrm{int}}}{d_{i(NR)\leftrightarrow \mathrm{int}} + d_{j(PSBA)\leftrightarrow \mathrm{int}}} \quad (2)$$

Electric potential energy as a function of ion migration was calculated in consecutive rounds. Each computation round starts by calculating the initial electric potential energy with the given *n* charges in natural rubber and in P(S-BA). One positive charge is randomly chosen and transferred from the low-ε NR cube to a vacant site in the high-ε P(S-BA) cube. Then, the electric potential energy is calculated again. This operation was repeated, yielding the change of electric potential energy of the system as a function of the number of charge transfer steps. The extent of charge migration was measured using the ratio *s*/*Q*, where *s* is the number of transferred ions and *Q* is the initial total charge of the cube with the lowest dielectric constant, that is the cube which is losing charges. The experiment was terminated when 70% of the positive charges migrated from the low-ε cube to the high-ε cube. Altogether, 120 independent experiments were performed and computed, resulting in 120 electric potential energy variations versus *s/Q* curves. Each one of these curves are shown in Figure S2a and c. Percent variation of each point was 17%. Standard deviation bars for each point were omitted for clarity in Figure 5, but it is presented in Figure S2.

Figures S2a and S2b show the individual curves and their average, assuming that the two cubes have the same dielectric constant. It was done to verify the effect (i) of simply spreading of charges over a larger volume. Figures S2c and S2d show the same data but assuming that the cubes have the

known dielectric constants for natural rubber and P(S-BA), respectively. It was done to verify (ii) the influence of dielectric constant difference in electric potential energy of the virtual object.

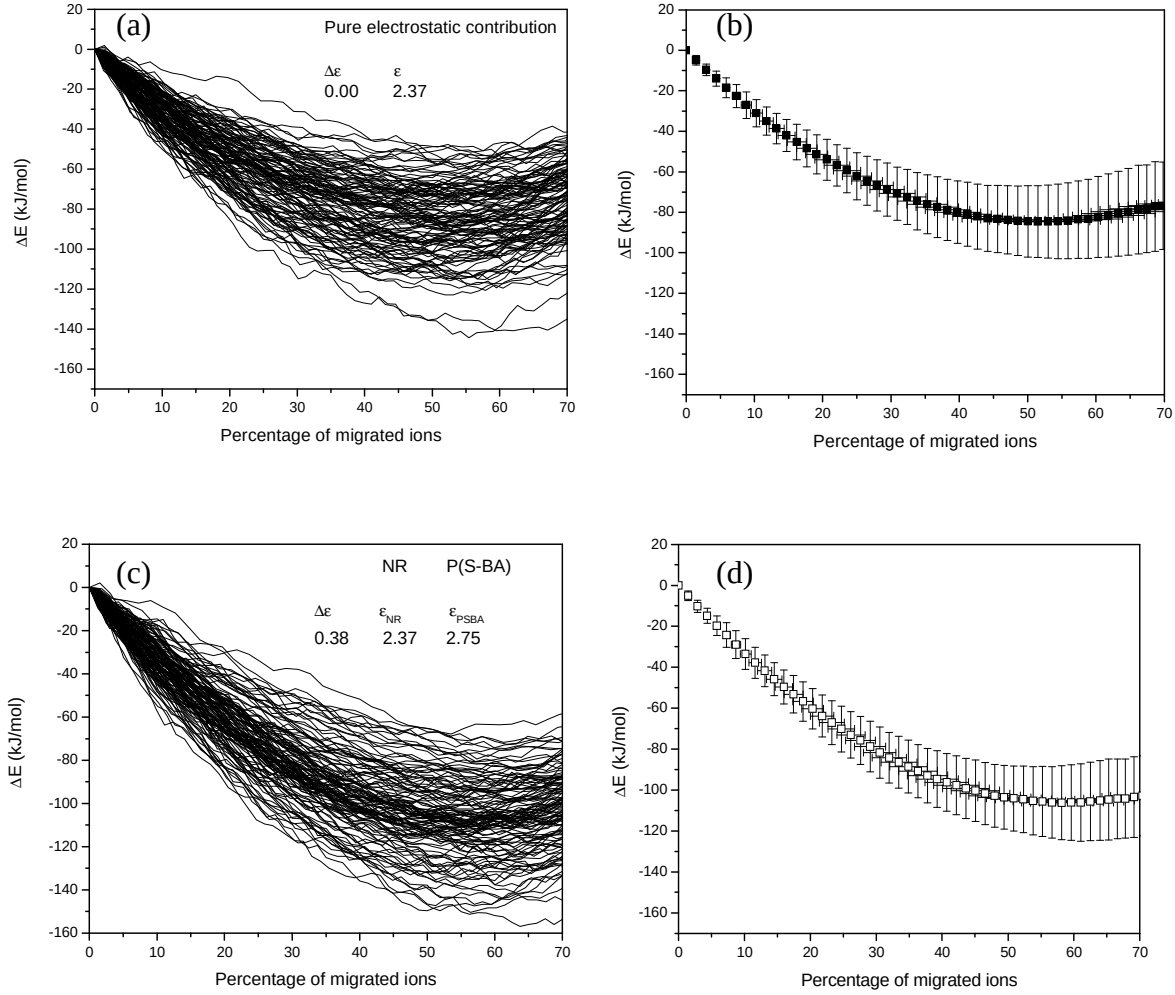


**Figure S2.** Individual plots of electrostatic energy change as a function of transferred ion fraction, from one to another cube of the virtual object (a, c). Each individual curve represents the migration process of each one of the 120 initial configurations considered. Average for all individual plots with error bars indicating the standard deviation of each migration event (b, d). $\Delta\varepsilon = 0.00$ in a and b. $\Delta\varepsilon = 0.38$ in c and d.

```
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

% PARTITION_CONTROLLER
% Program that controls Parition_core

% Cleanning home and introducing myself
clear all
fprintf('\n\n=== Electric charge partition in latex blends ===\n')

% Knowing the quantity of experiments to do
m = input('How many experiments do you want to do?    ');

v = input('If you continue a sequence of experiments, type the number of the last one. exp.#');v=round(v);

% Builting the cubes
n = input('Type the number of matrices that will form each cube:      n = ');
a = input('Type the distance between charges in METERS:               a = ');

% Defining characteristics of each polymer
epsilonM = input('Type the dielectric constant of material M:         epsilonM = ');
fM = input('Volume fraction of neutral species in M:                fM = ');
hM = input('Volume fraction of negative charges in M:               hM = ');

epsilonL = input('Type the dielectric constant of material L:         epsilonL = ');
fL = input('Volume fraction of neutral species in L:                fL = ');
hL = input('Volume fraction of negative charges in L:               hL = ');

% Defining the percentage of charges that will migrate to cube with higher
% epsilon. For instance, if w = 0.8 and epsilonM>epsilonL there will be
% migratin of 80% of positive charges of L to M.
w = input('What is the percentage of ions that will migrate to higher-epsilon cube? (e.g.: 0.8) \n');
```

```
% Executing the series of independent experiments

for experimentos = 1:1:m

    run Partition_core

    str1 = ['DeltaE_',int2str(experimentos+v),' = deltaE;',]; % gravando as matrizes
deltaE calculadas em cada experimento

    eval(str1);

    clear deltaE;

    str2 = ['Energia_',int2str(experimentos+v),' = energia;',]; % gravando as matrizes
energia calculadas em cada experimento

    eval(str2)

    clear energia

    cargas(experimentos,:) = [(experimentos+v) (QM/QL) QM qzM qnM qpM QL qzL qnL qpL ss];
% gravando as cargas iniciais em cada experimento

    str3                    =                    ['figure(',int2str(experimentos+v),');
plot(DeltaE_',int2str(experimentos+v),'(:,2),DeltaE_',int2str(experimentos+v),'(:,3))'];

    eval(str3);

    figure(experimentos+v); xlabel('Percentage of migrated cations'); ylabel('deltaE
(eV)');

end

clear m; clear str1; clear str2; clear str3;


% Presenting the calculation report

rel01 = ['REPORT'];

rel02 = ['   Number of experiments done           ',int2str(experimentos)];

rel03 = ['   Number of matrices in each cube      ',mat2str(n)];

rel04 = ['   Distance between charges             ',mat2str(a), ' m'];

rel05 = ['   Aresta of each cube                  ',mat2str((n-1)*a),' m'];

rel06 = ['   Volume of each cube                  ',mat2str(((n-1)^3)*(a^3)),' m³'];

rel07          =          ['DATA          OF          THE          MATERIALS','
Cube M','         Cube L'];

rel08          =          ['                                Dielectric          constant
',mat2str(epsilonM),'           ',mat2str(epsilonL)];

rel09   =   ['          Volume   fraction   of   neutral   species                (%)
',mat2str(fM),'          ',mat2str(fL)];

rel10   =   ['          Volume   fraction   of   negative   charges              (%)
',mat2str(hM),'             ',mat2str(hL)];

rel11   =   ['          Volume   fraction   of   positive   charges              (%)
',mat2str(1-fM-hM),'           ',mat2str(1-fL-hL)];

rel12 = ['CHARGE MIGRATION'];

rel13   =   ['       Percentage   of   charges   migrated   to   higher-epsilon   material
',mat2str(w*100),' %'];

relatorio                                                                       =
char(rel01,rel02,rel03,rel04,rel05,rel06,rel07,rel08,rel09,rel10,rel11,rel12,rel13);

disp(relatorio)

fprintf('INITIAL CHARGES IN EACH EXPERIMENT\n');

fprintf('    exp.#     QM/QL    QM        q neu M   q neg M   q pos M   QL        q neu L
```

```
q neg L   q pos L   #Migr.\n');
disp(cargas)

fprintf('\nRESULT LIST OF THE %i EXPERIMENTS\n',experimentos)

for x=1:1:experimentos
    fprintf('#%i\n',x+v);
    fprintf('      s       s*100/Q  E (eV)    EM (eV)   EL (eV)   EML (eV)\n');
    str1 = ['disp(Energia_',int2str(x+v),')'];
    eval(str1)
    fprintf('        s          s*100/Q   deltaE      deltaEM    deltaEL    deltaEML   (E-
E0)/E0\n');
    str2 = ['disp(DeltaE_',int2str(x+v),')'];
    eval(str2)
end; clear str1; clear str2;

% Builting the matrix that will be copied to Origin. It comprises
% DeltaE_*(:,2) on x axis and DeltaE_*(:,3) on y axis, i. e.,
%s/Q versus deltaE
final = 0.1234*ones(n^3,2*experimentos);
for c = 1:2:2*experimentos
    str1 = ['[b,x] = size(DeltaE_',int2str(v+(c+1)/2),');'];
    eval(str1);
    for l = 1:1:b
        str2                    =                   ['final(',int2str(l),',',int2str(c+v),')=
DeltaE_',int2str(v+(c+1)/2),'(',int2str(l),',2);'];
        eval(str2);
        str3                    =                   ['final(',int2str(l),',',int2str(c+1),')=
DeltaE_',int2str(v+(c+1)/2),'(',int2str(l),',3);'];
        eval(str3);
    end
end; clear c; clear l; clear x; clear str1; clear str2; clear str3; clear b;

% Builting another matrix to go to Origin. It comprises the number of
% migrated ions on the first column and values of deltaE on the following
% columns, for each experiment.
final2 = 0.1234*ones(n^3,experimentos);
for c = 1:1:experimentos
    str1 = ['[b,x] = size(DeltaE_',int2str(c+v),');'];
    eval(str1);
    for l = 1:1:b
        str2                    =                   ['final2(',int2str(l),',',int2str(c),')=
```

```
DeltaE_',int2str(c+v),'(',int2str(l),',3);'];

        eval(str2);

    end

end; clear c; clear l; clear x; clear str1; clear str2; clear str3; clear b;


% Builting another matrix to be plotted on Origin. It comprises

% Energia_*(:,2) on x axis and Energia_*(:,3) on y axis , i. e.,

% s*/Q versus E

final3 = 0.1234*ones(n^3,2*experimentos);

for c = 1:2:2*experimentos

    str1 = ['[b,x] = size(Energia_',int2str(v+(c+1)/2),');'];

    eval(str1);

    for l = 1:1:b

        str2                    =                    ['final3(',int2str(l),',',int2str(c),')=
Energia_',int2str(v+(c+1)/2),'(',int2str(l),',2);'];

        eval(str2);

        str3                    =                    ['final3(',int2str(l),',',int2str(c+1),')=
Energia_',int2str(v+(c+1)/2),'(',int2str(l),',3);'];

        eval(str3);

    end

end; clear c; clear l; clear x; clear str1; clear str2; clear str3; clear b;


% Evaluating the initial state: writting in a vector the absolute values

% with no migration executed. Exhibint the lower value.

for k = 1:1:experimentos

    str = ['Ezero(',int2str(k),',1) = Energia_',int2str(k+v),'(1,3);'];

    eval(str);

end; clear k; clear str;

fprintf('Number of experiments: %i\nMinimum energy: %f eV\nhM = %f\nhL =
%f\n',experimentos, min(Ezero),hM,hL)


% Cleanning home

clear rel01; clear rel02; clear rel03; clear rel04; clear rel05; clear rel06; clear
rel07;

clear rel08; clear rel09; clear rel10; clear rel11; clear rel12; clear rel13;clear E;
clear EM; clear EL; clear EML;

clear QM; clear QL; clear s; clear ss; clear x; clear qnM; clear qpM; clear qzM; clear
qnL; clear qpL; clear qzL;


% Farewell

fprintf('\nDone.\nProgram written by Sergio Jannuzzi on April 2009.\n\n')


=-=-=-=-=-=-=-=-=-=-=-
```

```
% PARTITION_CORE

% Presenting a new experiment
fprintf('\nBegin of the experiment %i.\n\n',(experimentos+v))

% Builting matrices

    % Matrices of cube M
QM = 0; qzM = 0; qpM = 0; qnM = 0;

for k = 1:1:n
    str = ['M',int2str(k),' = rand(n,n);'];
    eval(str);
        for x = 1:1:(n^2)
            str4 = ['M',int2str(k),'(x);'];
            if eval(str4)< fM
                str5 = ['M',int2str(k),'(x) = 0;'];
                eval(str5)
                qzM = qzM + 1;
            elseif eval(str4) > fM & eval(str4) < (fM+hM)
                str5 = ['M',int2str(k),'(x) = -1;'];
                eval(str5)
                qnM = qnM  + 1;
            elseif eval(str4) > (fM+hM)
                str5 = ['M',int2str(k),'(x) = +1;'];
                eval(str5)
                qpM = qpM + 1;
            end
        end
    str2 = ['M',int2str(k)];
    QM = QM + sum(sum(eval(str2)));
end
clear k; clear str; clear str2; clear str3; clear str4; clear str5; clear x;
fprintf('QM = %i\n',QM)

    % Matrices of cube L
QL = 0; qzL = 0; qpL = 0; qnL = 0;
```

```
for k = 1:1:n
    str = ['L',int2str(k),' = rand(n,n);'];
    eval(str);
        for x = 1:1:(n^2)
            str4 = ['L',int2str(k),'(x);'];
            if eval(str4)< fL
                str5 = ['L',int2str(k),'(x) = 0;'];
                eval(str5)
                qzL = qzL + 1;
            elseif eval(str4) > fL & eval(str4) < (fL+hL)
                str5 = ['L',int2str(k),'(x) = -1;'];
                eval(str5)
                qnL = qnL  + 1;
            elseif eval(str4) > (fL+hL)
                str5 = ['L',int2str(k),'(x) = +1;'];
                eval(str5)
                qpL = qpL + 1;
            end
        end
    str2 = ['L',int2str(k)];
    QL = QL + sum(sum(eval(str2)));
end
clear k; clear str; clear str2; clear str3; clear str4; clear str5; clear x;
fprintf('QL = %i\n',QL)

% Calculating the charge density in each cube
fprintf('Distance between charges: a = %g m\n',a)
rhoM = QM/(((n-1)^3)*(a^3));
fprintf('Charge density of polymer M: rhoM = %g unidades/m³\n',rhoM)
rhoL = QL/(((n-1)^3)*(a^3));
fprintf('Charge density of polymer L: rhoL = %g unidades/m³\n',rhoL)

% Defining the number of migrations
% ss is the number of migrations that will be done
if epsilonM>epsilonL
    ss = abs(round(qpL*w));
    Q = qpL;
else
    ss = abs(round(qpM*w));
```

```
    Q = qpM;
end


% Computing energy iteratively
s = 0;
run Particao_energy;


% Executing ionic migration
R = ceil(n*rand(1)); ii = ceil(n*rand(1)); jj = ceil(n*rand(1));
str1 = ['M',int2str(R),'(ii,jj)'];
S = ceil(n*rand(1)); kk = ceil(n*rand(1)); ll = ceil(n*rand(1));
str2 = ['L',int2str(S),'(kk,ll)'];


for s = 1:1:ss % performing ss ionic migrations
if epsilonM>epsilonL
    while (eval(str2) <= 0)
        S = ceil(n*rand(1)); kk = ceil(n*rand(1)); ll = ceil(n*rand(1)); % choosing randomly a non-zero charge in cube L
        str2 = ['L',int2str(S),'(kk,ll)'];
    end
    while (eval(str1) ~= 0)
        R = ceil(n*rand(1)); ii = ceil(n*rand(1)); jj = ceil(n*rand(1));% choosing randomly a vacant site in cube M (with 0)
        str1 = ['M',int2str(R),'(ii,jj)'];
    end
    tro = ['M',int2str(R),'(ii,jj) = L',int2str(S),'(kk,ll);']; % performing the migration
    eval(tro); % passing the charge from L to M
    trr = ['L',int2str(S),'(kk,ll) = 0;'];
    eval(trr);% migrating charge leaves a zero in its former place
else
    while (eval(str1) <= 0)
        R = ceil(n*rand(1)); ii = ceil(n*rand(1)); jj = ceil(n*rand(1)); % choosing randomly a non-zero charge in cube M
        str1 = ['M',int2str(R),'(ii,jj)'];
    end
    while (eval(str2) ~= 0)
        S = ceil(n*rand(1)); kk = ceil(n*rand(1)); ll = ceil(n*rand(1)); % choosing randomly a vacant site in cube M (with 0)
        str2 = ['L',int2str(S),'(kk,ll)'];
    end
    tro = ['L',int2str(S),'(kk,ll) = M',int2str(R),'(ii,jj);']; % performing the migration
```

```
    eval(tro); % passing the charge from M to L
    trr = ['M',int2str(R),'(ii,jj) = 0;'];
    eval(trr);% migrating charge leaves a zero in its former place
end
run Particao_energy;
end


% Cleanning home
clear str1; clear str2; clear g; ; clear R; clear S; clear epsilonzero; clear ii; clear jj; clear kk; clear ll; clear p; clear q; clear t; clear e;
clear trr; clear tro; clear epsilonML;


% Finishing the experiment
fprintf('\nEnd of the experiment %i.\n\n',(experimentos+v))


=-=-=-=-=-=-=-=-=-=


% PARTITION_ENERGY
% This part computes the energy of the system
E = 0; % This is the energy when charges are located infinitely away from each other. It will be calculated the energy to bring them together.
epsilonzero = 8.85e-12; %C²/N.m²
g = ((1.60217646e-19)^1)/(4*pi*epsilonzero*a); %constants


    % Computing energies insde cube M
p = 0; % p is just a counter of the number of non-zero iteractions. It must equals to x!/((x-2)!*2!) for x non-zero charges
for R = 1:1:n
    for S = R:1:n
        for ii = 1:1:n
            for jj = 1:1:n
                str1 = ['M',int2str(R),'(ii,jj)'];
                if eval(str1) ~= 0
                    for kk = 1:1:n
                        for ll = 1:1:n
                            str2 = ['M',int2str(S),'(kk,ll)'];
                            if R==S % i. e., if the evaluated charges are in the same matrix
                                if eval(str2) ~= 0
                                    if kk>ii
                                        e  =  g*eval(str1)*eval(str2)/(epsilonM*sqrt((ii-kk)^2 + (jj-ll)^2));
```

```
                                        E = E + e;
                                        p = p+1; %counting...
                                        %IM(p,:)=[R S ii jj kk ll e]; %this matrix writes
iterations inside cube M so that I know which calculations are being made
                                    elseif kk==ii & ll>jj
                                        e  =  g*eval(str1)*eval(str2)/(epsilonM*sqrt((ii-
kk)^2 + (jj-ll)^2));
                                        E = E + e;
                                        p = p+1; %counting...
                                        %IM(p,:)=[R S ii jj kk ll e]; %this matrix writes
iterations inside cube M so that I know which calculations are being made
                                    end
                                else
                                    e = 0;
                                    E = E + e;
                                end
                            else
                                if eval(str1) ~= 0
                                    if eval(str2) ~= 0
                                        e  =  g*eval(str1)*eval(str2)/(epsilonM*sqrt((ii-
kk)^2 + (jj-ll)^2) + (R-S)^2 );
                                        E = E + e;
                                        p = p+1; %counting...
                                        %IM(p,:)=[R S ii jj kk ll e]; %this matrix writes
iterations inside cube M so that I know which calculations are being made
                                    end
                                else
                                    e = 0;
                                    E = E +e;
                                end
                            end
                        end
                    end
                else
                    e = 0;
                    E = E + e;
                end
            end
        end
    end
end
EM = E;
```

```
    % Computing energies inside cube L

q = 0; % q is just a counter of the number of non-zero iteractions. It must equals to x!/((x-2)!*2!) for x non-zero charges

for R = 1:1:n

    for S = R:1:n

        for ii = 1:1:n

            for jj = 1:1:n

                str1 = ['L',int2str(R),'(ii,jj)'];

                if eval(str1) ~= 0

                    for kk = 1:1:n

                        for ll = 1:1:n

                            str2 = ['L',int2str(S),'(kk,ll)'];

                            if R==S

                                if eval(str2) ~= 0

                                    if kk>ii

                                        e  =  g*eval(str1)*eval(str2)/(epsilonL*sqrt((ii-kk)^2 + (jj-ll)^2));

                                        E = E + e;

                                        q = q+1; %counting...

                                        %IL(q,:)=[R  S  ii  jj  kk  ll  e]; % %this  matrix writes iterations inside cube L so that I know which calculations are being made

                                    elseif kk==ii & ll>jj

                                        e  =  g*eval(str1)*eval(str2)/(epsilonL*sqrt((ii-kk)^2 + (jj-ll)^2));

                                        E = E + e;

                                        q = q+1; %counting...

                                        %IL(q,:)=[R  S  ii  jj  kk  ll  e]; % %this  matrix writes iterations inside cube L so that I know which calculations are being made

                                    end

                                else

                                    e = 0;

                                    E = E + e;

                                end

                            else

                                if eval(str1) ~= 0

                                    if eval(str2) ~= 0

                                        e  =  g*eval(str1)*eval(str2)/(epsilonL*sqrt((ii-kk)^2 + (jj-ll)^2) + (R-S)^2 );

                                        E = E + e;

                                        q = q+1; %counting...

                                        %IL(q,:)=[R S ii jj kk ll e]; %this matrix writes iterations inside cube L so that I know which calculations are being made
```

```
                                end
                            else
                                e = 0;
                                E = E +e;
                            end
                        end
                    end
                else
                    e = 0;
                    E = E + e;
                end
            end
        end
    end
end
EL = E - EM;


    % Computing the energies between cube M and cube L
t = 0; % t is just a counter of the number of non-zero iteractions. It must equals to x!/((x-2)!*2!) for x non-zero charges
for R = 1:1:n
    for S = 1:1:n
        for ii = 1:1:n
            for jj = 1:1:n
                str1 = ['M',int2str(R),'(ii,jj)']; % index R runs the calculations in matrices of cube M
                if eval(str1) ~= 0
                    for kk = 1:1:n
                        for ll = 1:1:n
                            str2 = ['L',int2str(S),'(kk,ll)']; % index S runs calculation in matrices of cube L
                            if eval(str2) ~= 0
                                epsilonML = ((R-0.5)*epsilonM + (S-0.5)*epsilonL)/(R+S-1);
                                e = g*eval(str1)*eval(str2)/(epsilonML*sqrt((ii-kk)^2 + (jj-ll)^2 + (R+S-1)^2));
                                E = E + e;
                                t = t+1; %counting...
                                %IML(t,:)=[R S ii jj kk ll e]; % this
                                %matrix writes iteractions between matrix R
                                %(element ii,jj) of cube M and matrix S
```

```
                                %(element jj,kk) of cube L so that I know
                                %which calculations are bing made.
                            else
                                e = 0;
                                E = E + e;
                            end
                        end
                    end
                else
                    e = 0;
                    E = E + e;
                end
            end
        end
    end
end
EML = E - EM - EL;


% Registrering values of energy
energia((s+1),:) = [s (s*100/Q) E EM EL EML];
deltaE((s+1),:) = [s (s*100/Q) (E-energia(1,3)) (EM-energia(1,4)) (EL-energia(1,5)) (EML-
energia(1,6)) (E-energia(1,3))*100/energia(1,3)];
disp(s)
```