```
/****************************************** gating_rev2 ************************************/


//================================================== Support from Press et al., 1992
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#define NR_END 1
#define FREE_ARG char*
#define SQR(a) ((sqrarg=(a)) == 0.0 ? 0.0 : sqrarg*sqrarg)

void nrerror(char error_text[])/* Numerical Recipes standard error handler */{
        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}

double *vector(long nl, long nh)/* allocate a float vector with subscript range v[nl..nh] */{
        double *v, *t;
        int i;
        v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
        if (!v) nrerror("allocation failure in vector()");
        t=v-nl+NR_END;
        for(i=nl; i<=nh; i++) t[i]=0.0;
        return v-nl+NR_END;
}

float *fvector(long nl, long nh)/* allocate a float vector with subscript range v[nl..nh] */{
        float *v, *t;
        int i;
        v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
        if (!v) nrerror("allocation failure in vector()");
        t=v-nl+NR_END;
        for(i=nl; i<=nh; i++) t[i]=0.0;
        return v-nl+NR_END;
}

long *ivector(long nl, long nh)/* allocate a int vector with subscript range v[nl..nh] */{
        long *v, *t;
        int i;
        v=(long *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(long)));
        if (!v) nrerror("allocation failure in ivector()");
        t=v-nl+NR_END;
        for(i=nl; i<=nh; i++) t[i]=0.0;
        return v-nl+NR_END;
}

double **matrix(long nrl, long nrh, long ncl, long nch)/* allocate a float matrix with subscript range
m[nrl..nrh][ncl..nch] */{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        double **m;
        int j, z;
        /* allocate pointers to rows */
        m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;
        /* allocate rows and set pointers to them */
        m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;
        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;
        /* return pointer to array of pointers to rows */
        for(j=nrl; j<=nrh; j++) for(z=ncl; z<=nch; z++) m[j][z]=0.0;
        return m;
}

int **nmatrix(long nrl, long nrh, long ncl, long nch)/* allocate a float matrix with subscript range
m[nrl..nrh][ncl..nch] */{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        int **m;
        int j, z;
```

```
        /* allocate pointers to rows */
        m=(int **) malloc((size_t)((nrow+NR_END)*sizeof(int*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;
        /* allocate rows and set pointers to them */
        m[nrl]=(int *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(int)));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;
        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;
        /* return pointer to array of pointers to rows */
        for(j=nrl; j<=nrh; j++) for(z=ncl; z<=nch; z++) m[j][z]=0.0;
        return m;
}

void free_vector(double *v, long nl, long nh)/* free a float vector allocated with vector() */{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_fvector(float *v, long nl, long nh)/* free a float vector allocated with vector() */{
        free((FREE_ARG) (v+nl-NR_END));
}

void free_ivector(long *v, long nl, long nh)/* free a int vector allocated with vector() */{
        free((FREE_ARG) (v+nl-NR_END));
}



void free_matrix(double **m, long nrl, long nrh, long ncl, long nch)/* free a float matrix allocated by
matrix() */{
        free((FREE_ARG) (m[nrl]+ncl-NR_END));
        free((FREE_ARG) (m+nrl-NR_END));
}

void tridag(double a[], double b[], double c[], double r[], double u[],unsigned long n){
        long j;
        double bet,*gam;
        gam=vector(0,n-1);
        if (b[0] == 0.0) nrerror("Error 1 in tridag");

        u[0]=r[0]/(bet=b[0]);
        for (j=1;j<n;j++) {
                gam[j]=c[j-1]/bet;
                bet=b[j]-a[j]*gam[j];
                if (bet == 0.0) nrerror("Error 2 in tridag");
                u[j]=(r[j]-a[j]*u[j-1])/bet;
        }
        for (j=(n-2);j>=0;j--)
                u[j] -= gam[j+1]*u[j+1];
        free_vector(gam,0,n-1);
}


#define IM1 2147483563
#define IM2 2147483399
#define AM1 (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV1 (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

float ran2(long *idum)/*Long period (> 2   1018) random number generator of L'Ecuyer with Bays-Durham shu□e
and added safeguards. Returns a uniform random deviate between 0.0 and 1.0 (exclusive of
the endpoint values). Call with idum a negative integer to initialize; thereafter, do not alter
idum between successive deviates in a sequence. RNMX should approximate the largest floating
value that is less than 1.*/{
        int j;
```

```
        long k;
        static long idum2=123456789;
        static long iy=0;
        static long iv[NTAB];
        float temp;
        if (*idum <= 0) { //Initialize.
                if (-(*idum) < 1) *idum=1; //Be sure to prevent idum = 0.
                else *idum = -(*idum);
                idum2=(*idum);
                for (j=NTAB+7;j>=0;j--) { //Load the shuffle table (after 8 warm-ups).
                        k=(*idum)/IQ1;
                        *idum=IA1*(*idum-k*IQ1)-k*IR1;
                        if (*idum < 0) *idum += IM1;
                        if (j < NTAB) iv[j] = *idum;
                }
                iy=iv[0];
        }
        k=(*idum)/IQ1; //Start here when not initializing.
        *idum=IA1*(*idum-k*IQ1)-k*IR1; //Compute idum=(IA1*idum) % IM1 without
        if (*idum < 0) *idum += IM1; //overflows by Schrage's method.
        k=idum2/IQ2;
        idum2=IA2*(idum2-k*IQ2)-k*IR2; //Compute idum2=(IA2*idum) % IM2 likewise.
        if (idum2 < 0) idum2 += IM2;
        j=iy/NDIV1; //Will be in the range 0..NTAB-1.
        iy=iv[j]-idum2; //Here idum is shuffled, idum and idum2 are
        iv[j] = *idum; //combined to generate output.
        if (iy < 1) iy += IMM1;
        if ((temp=AM1*iy) > RNMX) return RNMX; //Because users don't expect endpoint values.
        else return temp;
}

double gasdev(long *idum)/*Returns a normally distributed deviate with zero mean and unit variance, using
ran1(idum)
as the source of uniform deviates.*/{
        double ran1(long *idum);
        static int iset=0;
        static float gset;
        double fac,rsq,v1,v2;
        if (*idum < 0) iset=0; //Reinitialize.
        if (iset == 0) { //We don't have an extra deviate handy, so
                do {
                        v1=2.0*ran2(idum)-1.0; //pick two uniform numbers in the square ex
                        v2=2.0*ran2(idum)-1.0; //tending from -1 to +1 in each direction,
                        rsq=v1*v1+v2*v2; //see if they are in the unit circle,
                }
                while (rsq >= 1.0 || rsq == 0.0); //and if they are not, try again.
                fac=sqrt(-2.0*log(rsq)/rsq);
                /*Now make the Box-Muller transformation to get two normal deviates. Return one and
                save the other for next time.*/
                gset=v1*fac;
                iset=1;
                //Set flag.
                return v2*fac;
        }
        else { //We have an extra deviate handy,
                iset=0; //so unset the flag,
                return gset; //and return it.
        }
}

//================================================================ end support

#define N 328
#define Npore 6
#define Lpore 0.2e-9
#define Latria 3.4e-9
#define thetaa 0.26
#define d0 1e-9
#define Pi 3.1415926535
#define Nions 2
double dt=0.01e-9;
double dtredLang=0.01e-9;
double dtred=2.0e-9;
#define F 96500.0
#define R 8.31
#define T 300.0
```

```
#define sigma 1.0e-10
#define eps0 8.85e-12
#define e0 1.6e-19
double xL=24e-10;
#define kB 1.38e-23
#define gamma 4e-6
#define ksp 0.0
#define xeqsp -1.3e-9

int Np, NS, iLang, iLangold;
double charge=0.0, chargeL=0.0, chargeR=0.0;
double Vm=-0.08, total_time=1e-3;
int i, j, NPL, NPR, NL, NAL, NAR, nS4init, nS4end, isfreport=1, istreport=1, assess_all_curr=0;
int n1S, Nml3, Nl3, ieqsp;
double *x, *Area, *Vol, *radius, dx, d1, d2, l0, l1, l2, l3;
double theta, erre, thetaold, erreold, *zions, **Dions, *C1, *C2, **cs,Vm;
double ttime,cumtime, *af, *bf, *cf, *rf, **a1f, **a2f, **c1f, **c2f, *zF_RT,*v, *v_old;
double **fluxes,*current, *s, *av, *bv,*cv, *rv, *eps, **aflux, **bflux;
double *aL, *bL,*cL,*rL, a1L, a2L, *fL, *muL, kT,*zS4, mufact, *zmeanS4;
double XLang, XLangold,Langfact,Langfactred, **intQ, **Qdispl, **intS4, *Gel;
double vLang,*currentLang, *currentDispl, *currentTot, **voltagemuL, springfact;
int move_S4=1; //boolean S4 S4 movement in go_Lang and go_Lang_reduced
double t_S4=0.0;
long idum,current_time;
FILE *dat, *tdat, *txtdat;

int x_to_n(double xx) {
        int ii=0;
        while(x[ii]<=xx) ii+=1;
        return(ii-1);
}

void setup_report(void) {
        if(isfreport==1) dat=fopen("F://TEMP//dati.asc","w");
        if(istreport==1) tdat=fopen("F://TEMP//tdati.asc","w");
        if(istreport==1) txtdat=fopen("F://TEMP//txtdati.asc","w");
}

void freport(void) {
        int ntemp1, ntemp2;
        double tempt;
        fprintf(dat,
"x\tArea\tVol\tradius\teps\tD0\tD1\tv\tcs0\tcs1\ts\tzS4\tzmeanS4\trv\tcurrent\tfluxes0\t");
        fprintf(dat, "fluxes1\tcurrentLang\tcurrentDispl\tcurrentTot\tfL\tmuL\tGel\n");

        for(i=0; i<=N-1; i++) {
                fprintf(dat, "%e\t%e\t%e\t%e\t%e\t%e\t%e", x[i], Area[i], Vol[i], radius[i], eps[i],
Dions[i][0], Dions[i][1]);
                fprintf(dat, "\t%e\t%e\t%e\t%e\t%e\t%e\t%e", v[i], cs[0][i], cs[1][i], s[i], zS4[i],
zmeanS4[i], rv[i]);
                fprintf(dat, "\t%e\t%e\t%e\t%e\t%e\t%e", current[i], fluxes[i][0], fluxes[i][1],
currentLang[i],
                        currentDispl[i], currentTot[i]);
                ntemp1=x_to_n(-xL/2);
                ntemp2=x_to_n(xL/2);
                if(i>=ntemp1+1 && i<ntemp2+2)                   {
                        fprintf(dat, "\t%e\t%e\t%e\n", fL[i-ntemp1-1], muL[i-ntemp1-1],
                                Gel[i-ntemp1-1]);
                }
                else {
                        fprintf(dat, "\t--\t--\t--\t--\t--\n");
                }
        }
        i=N;
        fprintf(dat, "%e\t%e\t-\t%e\n", x[i], Area[i],  radius[i]);
        fprintf(dat,"N=%d\n",N);
        fprintf(dat,"Npore=%d\n",Npore);
        fprintf(dat,"Lpore=%e\n",Lpore);
        fprintf(dat,"Latria=%e\n",Latria);
        fprintf(dat,"d0=%e\n",d0);
        fprintf(dat,"Nions=%d\n",Nions);
        fprintf(dat,"dt=%e\n",dt);
        fprintf(dat,"sigma=%e\n",sigma);
        fprintf(dat,"xL=%e\n",xL);
        fprintf(dat,"gamma=%e\n",gamma);
        fprintf(dat,"Np=%d\n",Np);
```

```
        fprintf(dat,"Vm=%e\n",Vm);
        fprintf(dat,"total_time=%e\n",total_time);
        fprintf(dat,"dx=%e\n",dx);
        fprintf(dat,"d1=%e\n",d1);
        fprintf(dat,"d2=%e\n",d2);
        fprintf(dat,"l1=%e\n",l1);
        fprintf(dat,"l2=%e\n",l2);
        fprintf(dat,"l3=%e\n",l3);
        fprintf(dat,"NPL=%d\n",NPL);
        fprintf(dat,"NPR=%d\n",NPR);
        fprintf(dat,"NAL=%d\n",NAL);
        fprintf(dat,"NAR=%d\n",NAR);
        fprintf(dat,"Nml3=%d\n",Nml3);
        fprintf(dat,"Nl3=%d\n",Nl3);
        fprintf(dat,"XLang=%e\n",XLang);
        for(i=0;i<Nions;i++) fprintf(dat,"zions=%e\n",zions[i]);
        for(i=0;i<Nions;i++) fprintf(dat,"C1=%e\n",C1[i]);
        for(i=0;i<Nions;i++) fprintf(dat,"C2=%e\n",C2[i]);
        fprintf(dat,"NL=%d\n",NL);
}

void build_geometry(void) {
        double temp1, temp2;
        dx=Lpore/Npore;
        d1=Lpore/2;
        d2=d1+Latria;
        l0=d1-d0*cos(thetaa)/(2.0*sin(thetaa));
        l1=l0+d0/(2.0*sin(thetaa));
        l2=l0+(d2-l0)/cos(thetaa);
        l3=d2+(d2-l0)*sin(thetaa)/cos(thetaa);
        i=N/2;
        x[i]=0.0;
        Area[i]=Pi*d0/2*d0/2;
        radius[i]=d0/2;
        i+=1;
        while((x[i-1]+dx)<=d1) {
                x[i]=x[i-1]+dx;
                radius[i]=d0/2;
                Area[i]=Area[i-1];
                Vol[i-1]=Area[i]*dx;
                i+=1;
        }
        x[i]=x[i-1]+dx;
        radius[i]=(x[i]-l0)*sin(thetaa)/cos(thetaa);
        thetaold=(thetaa)*((x[i]-d1)/(l1-d1));
        erreold=d0/(2.0*sin(thetaold));
        Area[i]=2.0*Pi*erreold*erreold*(1.0-cos(thetaold));
        Vol[i-1]=(2.0/3)*Pi*pow(erreold,3.0)*(1.0-cos(thetaold))-
            (Pi*(d0/2)*(d0/2)*erreold*cos(thetaold)/3.0)+Pi*(d0/2)*(d0/2)*(d1-x[i-1]);
        i+=1;
        while((x[i-1]+dx)<=l1) {
                x[i]=x[i-1]+dx;
                radius[i]=(x[i]-l0)*sin(thetaa)/cos(thetaa);
                theta=(thetaa)*((x[i]-d1)/(l1-d1));
                erre=d0/(2.0*sin(theta));
                Area[i]=2.0*Pi*erre*erre*(1.0-cos(theta));
                temp1=(2.0/3)*Pi*pow(erre,3.0)*(1.0-cos(theta))-Pi*(d0/2)*(d0/2)*erre*cos(theta)/3;

temp2=((2.0/3)*Pi*pow(erreold,3.0)*(1.0-cos(thetaold))-Pi*(d0/2)*(d0/2)*erreold*cos(thetaold)/3);
                Vol[i-1]=temp1-temp2;
                i+=1;
                erreold=erre;
                thetaold=theta;
        }
        x[i]=x[i-1]+dx;
        radius[i]=(x[i]-l0)*sin(thetaa)/cos(thetaa);
        theta=thetaa;
        erre=x[i]-l0;
        Area[i]=2.0*Pi*erre*erre*(1.0-cos(theta));
        temp1=(2.0/3)*Pi*pow(erre,3.0)*(1.0-cos(theta))-Pi*(d0/2)*(d0/2)*(d1-l0)/3;
        temp2=((2.0/3)*Pi*pow(erreold,3.0)*(1.0-cos(thetaold))-
                    Pi*(d0/2)*(d0/2)*erreold*cos(thetaold)/3);
        Vol[i-1]=temp1-temp2;
        i+=1;
        erreold=erre;
        thetaold=theta;
```

```
        while((x[i-1]+dx)<=l2) {
                x[i]=x[i-1]+dx;
                if(x[i]<=d2) radius[i]=(x[i]-l0)*sin(thetaa)/cos(thetaa);
                theta=(thetaa);
                erre=x[i]-l0;
                Area[i]=2.0*Pi*erre*erre*(1.0-cos(theta));
                Vol[i-1]=(2.0/3)*Pi*pow(erre,3.0)*(1.0-cos(theta))-
                    (2.0/3)*Pi*pow(erreold,3.0)*(1.0-cos(thetaold));
                i+=1;
                erreold=erre;
                thetaold=theta;
        }
        x[i]=x[i-1]+dx;
        theta=(Pi/2-thetaa)*(x[i]-l2)/(l3-l2)+thetaa;
        erre=(d2-l0)*sin(thetaa)/(cos(thetaa)*sin(theta));
        Area[i]=2.0*Pi*erre*erre*(1.0-cos(theta));

Vol[i-1]=(2.0/3)*Pi*pow(erre,3.0)*(1.0-cos(theta))-Pi*pow(((d2-l0)*sin(thetaa)/cos(thetaa)),2.0)*erre*cos(theta
)/3-

((2.0/3)*Pi*pow(l2-l0,3.0)*(1.0-cos(thetaa))-Pi*pow(((d2-l0)*sin(thetaa)/cos(thetaa)),2.0)*((d2-l0)/3))+
            (2.0/3)*Pi*pow(l2-l0,3.0)*(1.0-cos(thetaa))-(2.0/3)*Pi*pow(erreold,3.0)*(1.0-cos(thetaold));
        i+=1;
        erreold=erre;
        thetaold=theta;
        double tempt;
        while((x[i-1]+dx)<=l3) {
                x[i]=x[i-1]+dx;
                theta=(Pi/2-thetaa)*(x[i]-l2)/(l3-l2)+thetaa;
                erre=(d2-l0)*sin(thetaa)/(cos(thetaa)*sin(theta));
                Area[i]=2.0*Pi*erre*erre*(1.0-cos(theta));

temp1=(2.0/3)*Pi*pow(erre,3.0)*(1.0-cos(theta))-Pi*pow((d2-l0)*sin(thetaa)/cos(thetaa),2.0)*erre*cos(theta)/3;

temp2=((2.0/3)*Pi*pow(erreold,3.0)*(1.0-cos(thetaold))-Pi*pow((d2-l0)*sin(thetaa)/cos(thetaa),2.0)*erreold*cos(
thetaold)/3);
                Vol[i-1]=temp1-temp2;
                i+=1;
                erreold=erre;
                thetaold=theta;
        }
        x[i]=x[i-1]+dx;
        erre=x[i]-d2;
        Area[i]=2.0*Pi*erre*erre;
        Vol[i-1]=(2.0/3)*Pi*pow(erre,3.0)-((2.0/3)*Pi*pow(erreold,3.0)*(1.0-cos(thetaold))-
                    Pi*pow((d2-l0)*sin(thetaa)/cos(thetaa),2.0)*erreold*cos(thetaold)/3);
        i+=1;
        erreold=erre;
        thetaold=theta;
        while(x[i-1]<=4.4e-9) {
                x[i]=x[i-1]+dx;
                erre=(x[i]-d2);
                Area[i]=2.0*Pi*erre*erre;
                Vol[i-1]=(2.0/3)*Pi*(pow(erre,3.0)-pow(erreold,3.0));
                i+=1;
                erreold=erre;
                thetaold=theta;
        }
        while(i<=N) {
                x[i]=x[i-1]+2.0*(x[i-1]-x[i-2]);
                erre=(x[i]-d2);
                Area[i]=2.0*Pi*erre*erre;
                Vol[i-1]=(2.0/3)*Pi*(pow(erre,3.0)-pow(erreold,3.0));
                i+=1;
                erreold=erre;
                thetaold=theta;
        }
        for(i=N/2+1; i<=N;i++) x[N-i]=-x[i];
        for(i=N/2+1; i<=N;i++) radius[N-i]=radius[i];
        for(i=N/2+1; i<=N;i++) Area[N-i]=Area[i];
        for(i=N/2; i<=N-1;i++) Vol[N-i-1]=Vol[i];
        NPL=x_to_n(-Lpore/2);
        NPR=x_to_n(Lpore/2);
        NAL=x_to_n(-Lpore/2-Latria);
        NAR=x_to_n(Lpore/2+Latria);
        Nml3=x_to_n(-l3);
```

```
        Nl3=x_to_n(l3);
}

void setup(void) {
        int nsigma;
        x=vector(0,N);
        Area=vector(0,N);
        Vol=vector(0,N-1);
        radius=vector(0,N);
        build_geometry();
        zions=vector(0,Nions);
        Dions=matrix(0,N-1,0,Nions);
        C1=vector(0,Nions);
        C2=vector(0,Nions);
        cs=matrix(0,Nions, 0,N-1);
        af=vector(0,N-1);
        bf=vector(0,N-1);
        cf=vector(0,N-1);
        rf=vector(0,N-1);
        a1f=matrix(0,N-1,0,Nions);
        a2f=matrix(0,N-1,0,Nions);
        c1f=matrix(0,N-1,0,Nions);
        c2f=matrix(0,N-1,0,Nions);
        zF_RT=vector(0,Nions);
        v=vector(0,N-1);
        v_old=vector(0,N-1);
        fluxes=matrix(0,N-1,0,Nions);
        current=vector(0,N-1);
        currentLang=vector(0,N-1);
        currentDispl=vector(0,N-1);
        currentTot=vector(0,N-1);
        s=vector(0,N-1);
        av=vector(0,N-1);
        bv=vector(0,N-1);
        cv=vector(0,N-1);
        rv=vector(0,N-1);
        eps=vector(0,N-1);
        zS4=vector(0,N-1);

        //Shaker; Fixed with Y coordinate of Calfa,S4 3D distances from CZ phe to charge
        xL=36e-10;
        #define NS 31
        #define Np 6
        float nS[NS]={19.256, -17.56599, -13.759, -17.36499, -11.429, 16.58, 23.757, 23.304, 30.616,
                19.989, 17.49901, 17.99701, 19.961, 17.869, 19.625, 19.45,      9.98201, -4.892, -9.89499,
-14.248, -12.16699,
                -12.15799, -3.51, 17.17801, 20.341, 22.612, 25.787, 21.769, 18.929, 16.866, -18.5       };
        int zS[NS]={1,1,-1,-1,1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,1,-1,1, 1   };
        float xxp[Np]={-25.4, -17.9, -8.0, -3.4, 3.4, 10.0      };
        nsigma=(int) (sigma/dx);
        for(j=0; j<NS; j++) { n1S=x_to_n(nS[j]*1e-10);
                for(i=0;i<N;i++)
s[i]+=(1.0/sqrt(2.0*Pi*nsigma*nsigma))*exp(-1.0*(i-n1S)*(i-n1S)/(2.0*nsigma*nsigma))*zS[j];
        }
        double mup, mup2, ii,dd;
        for(j=0;j<=Np-1;j++) { mup=(double) x_to_n(xxp[j]*1e-10);
                for(i=0;i<=N-1;i++) { ii=(double) i;

zS4[i]+=(1.0/sqrt(2.0*Pi*nsigma*nsigma))*exp(-(1.0*(mup-ii)*(mup-ii))/(2.0*nsigma*nsigma));
                }
        }
        NL=(int) (xL/dx);
        if(NL % 2==1) NL+=1;
        fL=vector(0,NL-1);
        muL=vector(0,NL-1);
        intQ=matrix(0,NL-1,0, N-1);
        intS4=matrix(0,NL-1,0, N-1);
        Qdispl=matrix(0,NL-1,0, N-1);
        voltagemuL=matrix(0,NL-1,0, N-1);
        Gel=vector(0,NL-1);
        aL=vector(0,NL-1);
        bL=vector(0,NL-1);
        cL=vector(0,NL-1);
        rL=vector(0,NL-1);
        zmeanS4=vector(0,N-1);
        aflux=matrix(0,N-1,0,Nions);
```

```
        bflux=matrix(0,N-1,0,Nions);

        // parameter setting
        zions[0]=1.0;
        zions[1]=-1.0;
        for(i=0; i<Nions;i++) zF_RT[i]= F*zions[i]/(R*T);
        for(i=0; i<N;i++) Dions[i][0]=2e-10;
        for(i=0; i<N;i++) Dions[i][1]=2e-10;
        //ioni permeabili per l'intero percorso
        for(i=NPL; i<=NPR;i++) Dions[i][0]=0.0;
        for(i=NPL; i<=NPR;i++) Dions[i][1]=0.0;
        C1[0]=C2[0]=C1[1]=C2[1]=0.14;
        for(i=0;i<=N-1;i++) for(j=0; j<Nions;j++) cs[j][i]=C1[j]-i*(C1[j]-C2[j])/(N-1);
        for(i=NPL;i<=NPR-1;i++) for(j=0; j<Nions;j++) cs[j][i]=0.0;
        for(i=0; i<NAL;i++) v[i]=v_old[i]=Vm;
        for(i=NAL; i<=NAR;i++) v[i]=v_old[i]=Vm-Vm*(i-NAL)/(NAR-NAL);
        for(i=NAR+1; i<N;i++) v[i]=v_old[i]=0.0;
        fL[NL/2]=1.0;
        nS4init=0;
        while(zS4[nS4init]<=1e-8) nS4init+=1;
        nS4end=N-1;
        while(zS4[nS4end]<=1e-8) nS4end-=1;
        for(i=1; i<N-1;i++)     for(j=0;j<Nions;j++)
a1f[i][j]=2.0*Dions[i][j]*Area[i]*dt/((x[i+1]-x[i-1])*Vol[i]);
        for(i=1; i<N-1;i++)     for(j=0;j<Nions;j++)
a2f[i][j]=-zF_RT[j]*Dions[i][j]*Area[i]*dt/((x[i+1]-x[i-1])*Vol[i]);
        for(i=1; i<N-1;i++)     for(j=0;j<Nions;j++)
c1f[i][j]=2.0*Dions[i+1][j]*Area[i+1]*dt/((x[i+2]-x[i])*Vol[i]);
        for(i=1; i<N-1;i++)     for(j=0;j<Nions;j++)
c2f[i][j]=zF_RT[j]*Dions[i+1][j]*Area[i+1]*dt/((x[i+2]-x[i])*Vol[i]);
        bf[0]=1.0;
        cf[0]=0.0;
        bf[N-1]=1.0;
        af[N-1]=0.0;
        for(i=0; i<N;i++) eps[i]=80.0;
        for(i=NPL; i<NPR;i++) eps[i]=4.0;
        for(i=1;i<=N-1;i++) av[i]=-2.0*Area[i]*eps[i]*eps0/((x[i+1]-x[i-1])*Vol[i]);
        av[N-1]=0.0;
        for(i=0;i<=N-2;i++) cv[i]=-2.0*Area[i+1]*eps[i+1]*eps0/((x[i+2]-x[i])*Vol[i]);
        cv[0]=0.0;
        for(i=1;i<=N-2;i++) bv[i]=-(av[i]+cv[i]);
        bv[0]=1.0;
        bv[N-1]=1.0;
        kT=kB*T;
        a1L=dtred/(2.0*dx);
        a2L=kT*dtred/(gamma*dx*dx);
        mufact=-e0/(gamma*2.0*dx);
        for(i=1;i<=NL-2;i++) bL[i]=-1.0-2.0*a2L;
        for(i=1; i<N-1;i++)     for(j=0;j<Nions;j++) aflux[i][j]=-1e3*Area[i]*Dions[i][j]*2/(x[i+1]-x[i-1]);
        for(i=1; i<N-1;i++)     for(j=0;j<Nions;j++)
bflux[i][j]=-1e3*Area[i]*Dions[i][j]*zF_RT[j]/(x[i+1]-x[i-1]);
        ttime=0.0;
        cumtime=0.0;
        XLang=XLangold=0.0;
        Langfact=sqrt(2.0*kT*dt/gamma);
        Langfactred=sqrt(2.0*kT*dtredLang/gamma);
        iLang=iLangold=N/2;
        idum=-254;
        time(&current_time);
        idum=-current_time;
        ieqsp=x_to_n(xeqsp)-(N/2-NL/2);
        printf("\n\neeqsp=%d", ieqsp);
        springfact=-ksp*dx/gamma;
}

void finalize(void) {
        free_vector(x,0,N);
        free_vector(Area,0,N);
        free_vector(Vol,0,N-1);
        free_vector(radius, 0,N);
        free_vector(zions,0,Nions);
        free_matrix(Dions, 0,N-1,0,Nions);
        free_vector(C1,0,Nions);
        free_vector(C2,0,Nions);
        free_matrix(cs,0,Nions, 0,N-1);
        free_vector(af,0,N-1);
```

```
        free_vector(bf,0,N-1);
        free_vector(cf,0,N-1);
        free_vector(rf,0,N-1);
        free_matrix(a1f,0,N-1,0,Nions);
        free_matrix(a2f,0,N-1,0,Nions);
        free_matrix(c1f,0,N-1,0,Nions);
        free_matrix(c2f,0,N-1,0,Nions);
        free_vector(v,0,N-1);
        free_matrix(fluxes,0,N-1,0,Nions);
        free_vector(current,0,N-1);
        free_vector(currentLang,0,N-1);
        free_vector(s,0,N-1);
        free_vector(av,0,N-1);
        free_vector(bv,0,N-1);
        free_vector(cv,0,N-1);
        free_vector(rv,0,N-1);
        free_vector(eps,0,N-1);
        free_vector(fL,0,NL-1);
        free_vector(muL,0,NL-1);
        free_matrix(intQ,0,NL-1, 0, N-1);
        free_vector(Gel,0,NL-1);
        free_vector(aL,0,NL-1);
        free_vector(bL,0,NL-1);
        free_vector(cL,0,NL-1);
        printf("\nsaving data........");
}

void finalize_report(void) {
        if(isfreport==1) freport();
        if(isfreport==1) fclose(dat);
        if(isfreport==1) fclose(txtdat);
        if(istreport==1) fclose(tdat);
}

double show_time=1e-6;

/************************************************ go_Lang*************************************************
FULL SIMULATION, whithout approximations
requires: assess_cs() -> assess electrolyte ions concentration profiles from Nernst-Plank equation
assess_v()-> assess the electric potential profile by considering all fixed and mobile charges of the system
assess_Lang-> moves the s4 segment in accordance with the Langevin equation
assess_Fluxes -> assess electrolyte ions fluxed from the Nernst Plank equation

*/

//assess_cs ------ required
void assess_cs(void) {
        for(j=0;j<Nions;j++) {
                for(i=1;i<N-1;i++) af[i]=a1f[i][j]+a2f[i][j]*(v[i]-v[i-1]);
                for(i=1;i<N-1;i++) cf[i]=c1f[i][j]+c2f[i][j]*(v[i+1]-v[i]);
                for(i=1;i<N-1;i++)
                        bf[i]=-a1f[i][j]+a2f[i][j]*(v[i]-v[i-1])-c1f[i][j]+c2f[i][j]*(v[i+1]-v[i])-1.0;
                for(i=1;i<N-1;i++) rf[i]=-cs[j][i];
                rf[0]=C1[j];
                rf[N-1]=C2[j];

                tridag(af,bf,cf,rf,cs[j],N);
        }
}

// assess_v() ------------ required
void assess_v(void) {
        double temp=0;
        for(i=0; i<=N-1;i++) zmeanS4[i]=0.0;
        for(i=0; i<=N-1;i++) v_old[i]=v[i];
        for(i=nS4init;i<=nS4end;i++) zmeanS4[i-(N/2-iLang)]+=zS4[i];

        for(i=1;i<N-2;i++) {
                temp=0.0;
                for(j=0;j<Nions;j++) temp+=zions[j]*cs[j][i];
                rv[i]=1e3*temp*F+e0*(s[i]+zmeanS4[i])/Vol[i];
        }
        rv[0]=Vm;
        rv[N-1]=0.0;
        tridag(av,bv,cv,rv,v,N);
}
```

```
// assess_Lang() --------------- required

void assess_Lang(void) {
        double muLang,dxLang;

        muLang=0.0;
        for(j=(nS4init-(N/2-iLang));j<=(nS4end-(N/2-iLang));j++) muLang+=(v[j+1]-v[j-1])*zmeanS4[j];
        muLang=muLang*mufact;
        dxLang=muLang*dt+Langfact*gasdev(&idum);
        XLang=XLangold+dxLang;

        if(XLang>xL/2) XLang=xL/2-(XLang-xL/2);
        if(XLang<-xL/2) XLang=-xL/2+(-xL/2-XLang);
        iLang=x_to_n(XLang);
        vLang=dx*(iLang-iLangold)/dt;
        XLangold=XLang;
        iLangold=iLang;
}

// assess_fluxes() ------------- required

void assess_fluxes(void){
        double temp;
        for(i=1;i<=N-2;i++)
                for(j=0;j<Nions;j++)
                        fluxes[i][j]=
                            aflux[i][j]*(cs[j][i]-cs[j][i-1])+bflux[i][j]*(cs[j][i]+cs[j][i-1])*(v[i]-v[i-1]);
        for(i=1;i<=N-2;i++) {
                temp=0.0;
                for(j=0;j<Nions;j++) temp+=zions[j]*fluxes[i][j];
                current[i]=temp*F;
                currentLang[i]=0.5*(zmeanS4[i]/Vol[i]+zmeanS4[i-1]/Vol[i-1])*e0*vLang*Area[i];
        }
        for(i=1;i<=N-2;i++)
currentDispl[i]=2.0*eps0*eps[i]*Area[i]*(v[i-1]-v[i]-v_old[i-1]+v_old[i])/(dt*(x[i+1]-x[i-1]));
        for(i=1;i<=N-2;i++) currentTot[i]=current[i]+currentLang[i]+currentDispl[i];

}

void go_Lang(void){
        double tempshow;
        tempshow=show_time;
        ttime=0.0;
        if(istreport==1) fprintf(tdat,
"cumtime\tv[N/2]\tXLang\tcurrent[1]\tcurrent[N-1-1]\tcharge\tcurrentLang[N/2]");
        if(istreport==1) fprintf(tdat, "\tcurrentDispl[N/2]\tcurrentTot[N/2]\n");

        while(ttime<total_time) {
                if(move_S4==1) {
                        assess_Lang();
                }
                else {
                        if (ttime>=t_S4) {
                                XLang=dx;
                                iLang=x_to_n(XLang);
                                vLang=dx*(iLang-iLangold)/dt;
                                XLangold=XLang;
                                iLangold=iLang;
                        }
                }
                assess_v();
                assess_cs();
                assess_fluxes();
                charge+=dt*(current[1]+current[N-1-1])/2.0;
                ttime+=dt;
                cumtime+=dt;
                if(ttime>=tempshow) {
                        printf("\n%e\t%e", ttime, XLang);
                        if(istreport==1) fprintf(tdat, "%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\n", cumtime, v[N/2],
XLang,current[1],
                                current[N-1-1], charge, currentLang[N/2], currentDispl[N/2], currentTot[N/2]);
                        tempshow+=show_time;
                }
        }
```

```
}

/************************************************
go_Lang_reduced*************************************************
REDUCED MODEL SIMULATION, with approximations
requires: assess_muL() -> assess mu for each sensor position
assess_v()-> assess the electric potential profile by considering all fixed and mobile charges of the system
assess_Lang-> moves the s4 segment in accordance with the Langevin equation
assess_Fluxes -> assess electrolyte ions fluxed from the Nernst Plank equation

*/

// go_Lang_eq -------------------- required ------------------- required

void go_Lang_eq(int iL) {
        iLang=iL;
        XLang=x[iL];
        double err;
        double tempshow, show_time=1e-7;
        double *tempv;
        tempv=vector(0,N-1);
        err=1.0;
        while(err>1e-6) {
                assess_cs();
                assess_v();
                err=0.0;
                for(i=1;i<N-1;i++) tempv[i]=fabs(v_old[i]-v[i]);
                for(i=1;i<=N-2;i++) if(tempv[i]>err) err=tempv[i];
        }

}

// assess_mul() -------------------- required
// requires go_Lang_eq() assess equilibrium for a sensor position
void assess_muL(void) {
        int imuL, iions;
        double temp, temp2, temp3;
        for(imuL=0;imuL<=NL-1;imuL++) {
                go_Lang_eq(N/2-NL/2+imuL);
                muL[imuL]=0.0;
                for(j=NAL;j<=NAR;j++) muL[imuL]+=(v[j-(NL/2-imuL)+1]-v[j-(NL/2-imuL)-1])*zS4[j];
                muL[imuL]=muL[imuL]*mufact;
                muL[imuL]+=springfact*(imuL-ieqsp);
                printf("\ni=%d, mu=%e",imuL,muL[imuL]);
                for(i=0;i<=N-1;i++) voltagemuL[imuL][i]=v[i];
                for(i=0;i<=N-1;i++) intQ[imuL][i]=0.0;
                for(iions=0; iions<Nions;iions++) {
                        temp=cs[iions][NPL-1];
                        temp2=Vol[NPL-1];
                        intQ[imuL][NPL-1]+= zions[iions]*temp*temp2*1e3*F;
                }
                for(j=NPL-2;j>=0;j--) {
                        intQ[imuL][j]=intQ[imuL][j+1];
                        for(iions=0; iions<Nions;iions++)
intQ[imuL][j]+=zions[iions]*cs[iions][j]*Vol[j]*1e3*F;
                }
                for(iions=0; iions<Nions;iions++) {
                        temp=cs[iions][NPR+1];
                        temp2=Vol[NPR+1];
                        intQ[imuL][NPR+1]-= zions[iions]*temp*temp2*1e3*F;
                }
                for(j=NPR+2;j<=N-1;j++) {
                        intQ[imuL][j]=intQ[imuL][j-1];
                        for(iions=0; iions<Nions;iions++)
intQ[imuL][j]-=zions[iions]*cs[iions][j]*Vol[j]*1e3*F;
                }
                for(i=0;i<=N-1;i++) intS4[imuL][i]=0.0;
                intS4[imuL][N-1]=zmeanS4[N-1]*e0;
                for(j=N-2;j>=0; j--) intS4[imuL][j]=intS4[imuL][j+1]+zmeanS4[j]*e0;
                for(i=1;i<=N-2;i++) Qdispl[imuL][i]=2.0*eps[i]*eps0*Area[i]*(v[i-1]-v[i])/(x[i+1]-x[i-1]);
                Gel[imuL]=0.0;
                for(j=NAL;j<=NAR;j++) Gel[imuL]+=v[j-(NL/2-imuL)]*zS4[j]*e0;
        }
}

// assess_Lang_reduced() ---------------------- required
```

```
void assess_Lang_reduced(void) {
        double dxLang;
        dxLang=muL[-(N/2-NL/2)+iLang]*dtredLang+Langfactred*gasdev(&idum);
        XLang=XLangold+dxLang;
        if(XLang>xL/2) XLang=xL/2-(XLang-xL/2);
        if(XLang<-xL/2) XLang=-xL/2+(-xL/2-XLang);
        iLang=x_to_n(XLang);
        vLang=dx*(iLang-iLangold)/dtredLang;
        iLangold=iLang;
        XLangold=XLang;
}

void go_Lang_reduced(void) {
        double IgoldL, IgL, IgoldR, IgR, *Q,*Qold, *Qdd, *Qddold,*QS4, *QS4old;
        double tempshow;
        tempshow=show_time;
        Qold=vector(0,N-1);
        Q=vector(0,N-1);
        Qdd=vector(0,N-1);
        Qddold=vector(0,N-1);
        QS4old=vector(0,N-1);
        QS4=vector(0,N-1);
        ttime=0.0;
        assess_muL();
        XLang=XLangold=0.0;
        iLang=iLangold=x_to_n(XLang);
        for(i=0;i<=N-1;i++) Qold[i]=intQ[-(N/2-NL/2)+iLang][i];
        for(i=0;i<=N-1;i++) Qddold[i]=Qdispl[-(N/2-NL/2)+iLang][i];
        for(i=0;i<=N-1;i++) QS4old[i]=intS4[-(N/2-NL/2)+iLang][i];
        if(istreport==1) fprintf(tdat, "cumtime\tcurrent[1]\tcurrent[N-1-1]\tcharge\n");
        while(ttime<total_time) {
                if(move_S4==1) {
                        assess_Lang_reduced();
                }
                else {
                        if (ttime>=t_S4) {
                                XLang=dx;
                                iLang=x_to_n(XLang);
                        }
                }
                ttime+=dtredLang;
                cumtime+=dtredLang;
                for(i=0;i<=N-1;i++) Q[i]=intQ[-(N/2-NL/2)+iLang][i];
                for(i=0;i<=N-1;i++) Qdd[i]=Qdispl[-(N/2-NL/2)+iLang][i];
                for(i=0;i<=N-1;i++) QS4[i]=intS4[-(N/2-NL/2)+iLang][i];
                //assess ionic currents
                for(i=0;i<=N-1;i++) current[i]=(Q[i]-Qold[i])/dtred;
                for(i=0;i<=N-1;i++) currentLang[i]=(QS4[i]-QS4old[i])/dtred;
                for(i=0;i<=N-1;i++) currentDispl[i]=(Qdd[i]-Qddold[i])/dtred;
                chargeL+=Q[0]-Qold[0];
                IgL=current[0];
                chargeR+=Q[N-1]-Qold[N-1];
                IgR=current[N-1];
                for(i=1;i<=N-2;i++) currentTot[i]=current[i]+currentLang[i]+currentDispl[i];
                for(i=0;i<=N-1;i++) Qold[i]=Q[i];
                for(i=0;i<=N-1;i++) Qddold[i]=Qdd[i];
                for(i=0;i<=N-1;i++) QS4old[i]=QS4[i];
                if(ttime>=tempshow) {
                        if(istreport==1) fprintf(tdat, "%e\t%e\t%e\n", cumtime, current[1], current[N-1-1]);
                        tempshow+=show_time;
                }
        }
}

void assess_FP(void) {
        double tempL;
        for(i=1;i<=NL-2;i++) aL[i]=a1L*muL[i-1]+a2L;
        for(i=1;i<=NL-2;i++) cL[i]=-a1L*muL[i+1]+a2L;
        for(i=0;i<=NL-1;i++) rL[i]=-fL[i];
        tempL=-1.0+muL[NL-1]*a1L-a2L;
        bL[NL-1]=tempL;
        aL[NL-1]=muL[NL-2]*a1L+a2L;
        bL[0]=-1.0-muL[0]*a1L-a2L;
        cL[0]=-muL[1]*a1L+a2L;
        tridag(aL,bL,cL,rL,fL,NL);
        tempL=0.0;
```

```
        for(i=0;i<=NL-1;i++) tempL+=fL[i];
        if(tempL<1.0) printf("\n\nWarning!! fL<1\n");
}

int assess_muL_tag=1;

void go_FP(void) {
        double tempshow, IgoldL, IgL, IgoldR, IgR, *Q,*Qold, *QS4,*QS4old, xmean, Fmean, Qpore, *Qdd, *Qddold;
        tempshow=show_time;
        ttime=0.0;
        Q=vector(0,N-1);
        Qold=vector(0,N-1);
        QS4=vector(0,N-1);
        QS4old=vector(0,N-1);
        Qdd=vector(0,N-1);
        Qddold=vector(0,N-1);
        for(j=0;j<N;j++) Qold[j]=0;
        for(j=0;j<N;j++) QS4old[j]=0;
        for(j=0;j<N;j++) Qddold[j]=0;
        if(assess_muL_tag==1) assess_muL();
        for(j=0;j<N;j++) for(i=0;i<NL;i++) Qold[j]+=intQ[i][j]*fL[i];
        for(j=0;j<N; j++) for(i=0;i<NL;i++) QS4old[j]+=intS4[i][j]*fL[i];
        for(j=0;j<N;j++) for(i=0;i<NL;i++) Qddold[j]+=Qdispl[i][j]*fL[i];
        while(ttime<total_time) {
                assess_FP();
                ttime+=dtred;
                cumtime+=dtred;
                if(ttime>=tempshow) {
                        if(assess_all_curr==1) {
                                for(j=0;j<N;j++) Q[j]=0;
                                for(j=0;j<N;j++) QS4[j]=0;
                                for(j=0;j<N;j++) Qdd[j]=0;
                                for(j=0;j<N;j++) for(i=0;i<NL;i++) Q[j]+=intQ[i][j]*fL[i];
                                for(j=0;j<N; j++) for(i=0;i<NL;i++) QS4[j]+=intS4[i][j]*fL[i];
                                for(j=0;j<N;j++) for(i=0;i<NL;i++) Qdd[j]+=Qdispl[i][j]*fL[i];
                                for(i=0;i<=N-1;i++) current[i]=(Q[i]-Qold[i])/show_time;
                                for(i=0;i<=N-1;i++) currentLang[i]=(QS4[i]-QS4old[i])/show_time;
                                for(i=0;i<=N-1;i++) currentDispl[i]=(Qdd[i]-Qddold[i])/show_time;
                                for(i=1;i<=N-2;i++) currentTot[i]=current[i]+currentLang[i]+currentDispl[i];
                                chargeL+=Q[0]-Qold[0];
                                chargeR+=Q[N-1]-Qold[N-1];
                                for(i=0;i<=N-1;i++) Qold[i]=Q[i];
                                for(i=0;i<=N-1;i++) QS4old[i]=QS4[i];
                                for(i=0;i<=N-1;i++) Qddold[i]=Qdd[i];
                                xmean=0.0;
                                for(i=0;i<NL;i++) xmean+=(-xL/2+dx*i+dx/2)*fL[i];
                                Fmean=0.0;
                                for(i=0;i<NL;i++) Fmean+=muL[i]*fL[i]*gamma;
                                Qpore=0.0;
                                for(i=0;i<NL;i++) {
                                        for(j=NPL; j<=NPR;j++) Qpore+=zS4[j+(NL/2-i)]*fL[i];
                                        if(istreport==1) fprintf(tdat, "%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\t%e\n",
cumtime, fL[NL-1], current[0]*1e12,
                                                        current[N-1]*1e12, chargeL/e0, chargeR/e0, xmean, Fmean,Qpore);
                                }
                        }
                        else {
                                Q[0]=0;
                                Q[N-1]=0;
                                for(i=0;i<NL;i++) Q[0]+=intQ[i][0]*fL[i];
                                for(i=0;i<NL;i++) Q[N-1]+=intQ[i][N-1]*fL[i];
                                current[0]=(Q[0]-Qold[0])/show_time;
                                current[N-1]=(Q[N-1]-Qold[N-1])/show_time;
                                chargeL+=Q[0]-Qold[0];
                                chargeR+=Q[N-1]-Qold[N-1];
                                Qold[0]=Q[0];
                                Qold[N-1]=Q[0];
                                if(istreport==1) fprintf(tdat, "%e\t%e\n", cumtime, current[0]*1e12);
                        }
                        tempshow+=show_time;
                }
        }
}

void go_FP_eq(void) {
        double tempshow, IgoldL, IgL, Igold, QL,QoldL, show_time_old;
```

```
        show_time_old=show_time;
        show_time=1e-4;
        tempshow=show_time;
        ttime=0.0;
        if(assess_muL_tag==1) assess_muL();
        QoldL=0.0;
        for(i=0;i<NL;i++) QoldL+=intQ[i][0]*fL[i];
        IgL=1.0;
        while(fabs(IgL)>1e-18) {
                assess_FP();
                ttime+=dtred;
                if(ttime>=tempshow) {
                        QL=0.0;
                        for(i=0;i<NL;i++) QL+=intQ[i][0]*fL[i];
                        IgL=(QL-QoldL)/show_time;
                        QoldL=QL;
                        printf("\n%e\t%e", ttime, fabs(IgL));
                        tempshow+=show_time;
                }
        }
        charge=chargeL=chargeR=0.0;
        show_time=show_time_old;
}

void compute_QV(void) {
        int iQV, i;
        double *vtemp, *fLtemp, Vp=-0.13, Vh=-0.14;
        double *muLVh;
        muLVh=vector(0,NL-1);
        FILE *QV;
        QV=fopen("F://TEMP//Boltzmann.asc","w");
        Vm=Vh;
        assess_muL();
        for(i=0; i<NL; i++) muLVh[i]=muL[i];
        for(iQV=1;iQV<=11;iQV++) {
                charge=0.0;
                Vp+=0.01;
                fprintf(tdat, "VP=%e\n", Vp);
                Vm=Vh;
                assess_muL_tag=0;
                go_FP_eq();
                total_time=0.5e-3;
                show_time=10e-6;
                go_FP();
                fprintf(QV, "%e\t%e", Vp, chargeL/e0);
                Vm=Vp;
                total_time=40e-3;
                show_time=10e-6;
                assess_muL_tag=1;
                go_FP();
                fprintf(QV, "\t%e\t", chargeL/e0);
                printf("\nVm=%e\tcharge=%e\n",Vm,  chargeL/e0);
                Vm=Vh;
                for(i=0; i<NL; i++) muL[i]=muLVh[i];
                total_time=40e-3;
                show_time=10e-6;
                assess_muL_tag=0;
                go_FP();
                fprintf(QV, "%e\n", chargeL/e0);
        }
        fclose(QV);
}

void compute_ColeMoore(void) {
        int iQV, i;
        double *vtemp, *fLtemp, Vp=-0.06, Vh=-0.14;
        double *muLVh, *muLm20;
        muLVh=vector(0,NL-1);
        muLm20=vector(0,NL-1);
        FILE *QV;
        QV=fopen("D://TEMP//Boltzmann.asc","w");
        Vm=-0.02;
        assess_muL();
        for(i=0; i<NL; i++) muLm20[i]=muL[i];
        Vm=Vh;
        assess_muL();
```

```
        for(i=0; i<NL; i++) muLVh[i]=muL[i];
        for(iQV=1;iQV<=1;iQV++) {
                charge=0.0;
                Vp+=0.01;
                fprintf(tdat, "VP=%e\n", Vp);
                Vm=Vh;
                assess_muL_tag=0;
                for(i=0; i<NL; i++) muL[i]=muLVh[i];
                go_FP_eq();
                total_time=0.5e-3;
                show_time=10e-6;
                go_FP();
                fprintf(QV, "%e\t%e", Vp, chargeL/e0);
                Vm=Vp;
                total_time=5e-3;
                show_time=10e-6;
                assess_muL_tag=1;
                go_FP();
                fprintf(QV, "\t%e\t", chargeL/e0);
                printf("\nVm=%e\tcharge=%e\n",Vm,  chargeL/e0);
                Vm=-0.02;
                for(i=0; i<NL; i++) muL[i]=muLm20[i];
                total_time=10e-3;
                show_time=10e-6;
                assess_muL_tag=0;
                go_FP();
                fprintf(QV, "%e\n", chargeL/e0);
        }
        fclose(QV);
}


void main(void) {
        printf("Hello!!");
        setup();
        printf("\nelectrode distance=%e", x[0]);
        printf("\nl3=%e", l3);
        isfreport=1;
        istreport=1;
        setup_report();
        assess_all_curr=0;
        Vm=-0.19;
        go_FP_eq();
        Vm=-0.19;
        total_time=1e-3;
        show_time=5e-6;
        go_FP();
        Vm=-0.04;
        total_time=15e-3;
        show_time=5e-6;
        go_FP();
        finalize_report();
        finalize();
}
```