```
/*----------------------------------------------------------------
*
* This code is based on the original UCLA model, and is modified
* by CIRCS group of Northeastern University.
*
* Contact Information:
*
* Center for interdisciplinary research on complex systems
* Departments of Physics, Northeastern University
*
* Alain Karma           a.karma (at) northeastern.edu
* Mingwang Zhong        mingwang.zhong (at) gmail.com
*
* The code was used to reproduce simulations in
* Transient outward K+ current (Ito) underlies the right ventricular
* initiation of polymorphic ventricular tachycardia in a transgenic
* rabbit model of long QT type 1, Bum-Rak Choi, Weiyan Li, Dmitry
* Terentyev, Anatoli Kabkov, Mingwang Zhong, Colin M Rees, Radmila
* Terentyeva, Tae Yun Kim, Zhilin Qu, Xuwen Peng, Alain Karma,
* and Gideon Koren (2018).
*---------------------------------------------------------------- */

// Information of original UCLA model:
/*-------------------- UCLA Model ver 1.00 ----------------------
*
* Contact Information
*
* Departments of Medicine (Cardiology)
* David Geffen School of Medicine at UCLA
*
* Daisuke Sato           dasato (at) mednet.ucla.edu
* Yohannes Shiferaw      yshiferaw (at) csun.edu
* James N Weiss          JWeiss (at) mednet.ucla.edu
*
* The code was used to produce simulations in
* A. Mahajan, Y. Shiferaw, D. Sato, A. Baher, R. Olcese, L.-H. Xie,
* M.-J. Yang, P.-S. Chen, J. G. Restrepo, A. Karma, A. Garfinkel,
* Z. Qu, and J. N. Weiss, A rabbit ventricular action potential model
* replicating cardiac dynamics at rapid heart rates, Biophysical Journal,
* 94 (2008), pp. 392¿410.
*---------------------------------------------------------------- */

#include "cell.h"
#define pow2(x) ((x)*(x))
// initial conditions
CCell::CCell(void) : y(new double[N]),

        xm(y[0]), xh(y[1]), xj(y[2]), xhl(y[3]),
        IKrC1(y[4]), IKrC2(y[5]), IKrC3(y[6]), IKrO(y[7]), IKrI(y[8]),
        xs1(y[9]), xs2(y[10]),
        ica(y[11]), hd(y[12]), hf(y[13]), hf_ca(y[14]),
        xtos(y[15]), ytos(y[16]), xtof(y[17]), ytof(y[18]),
        step(y[19]), v(y[20]),
        cp(y[21]), cs(y[22]), ci(y[23]), cj(y[24]), cjp(y[25]),
        tropi(y[26]), trops(y[27]),
        jrel(y[28]), xir(y[29]), fspark(y[30]),
        xnai(y[31])
{
        // INa
        xm = 0.001145222753;// sodium m-gate
        xh = 0.9898351676;// sodium h-gate
        xj = 0.9930817518;// soiumj-gate

        // IKr
        IKrC1 = 1;// ikr gate variable
        IKrC2 = 0;
        IKrC3 = 0;
        IKrO = 0;
        IKrI = 0;

        // IKs
        xs1 = 0.08433669901;// iks gate variable
        xs2 = 0.1412866149;// iks gate varaible
```

```
        // ICa
        ica = -0.1; // single channel flux
        hd =0;
        hf =1;
        hf_ca =0;

        // Ito
        xtos = 0.01;// ito slow activation
        ytos = 0.667;// ito slow inactivation
        xtof = 0.003737842131;// ito fast activation
        ytof = 0.9823715315;// ito slow inactivation

        // other
        step = 0;
        dt = 0.0025; // time step
        v = -86.79545769; // voltage
        vold = v;

        cp = 0.182601371;// averaged dyadic space con.
        cs = 0.1205609256;// averaged submembrane conc.
        ci = 0.0863687451;// myoplasm conc.
        cj = 95; //107.0388739;// NSR load
        cjp = 90; //95.76256179;// average JSR load

        tropi = 29.64807803;// time dependent buffers in myplasm (troponin)
        trops = 26.37726416;// time dependent buffers in submembrane (troponin)

        jrel = 0;
        xir = 0.006462569526;// SR current flux
        fspark = 1.0;

        xnai = 5.1;// internal Na conc.

        #ifdef ___USE_VAR_FOR_CONST
                xnao = 136.0;//mM        external Na
                xki = 140.0;// mM         internal K
                xko = 5.4;//5.40;//mM     external K
                cao = 1.0;//1.8;// mM     external Ca
                ek = (1.0/frt)*log(xko/xki);

                gca = 70;// ical conductance
                gtos = 0.05;// ito slow conductance // 0.05 for RV, 0.025 for LV
                gtof = 0.08;// ito fast conductance
                gnaca = 0.756;// exchanger strength
                gkr = 0.096;// Ikr conductance, MGWMN: 0.096,  HH: 0.16
                gks = 0; // LMC: RV 0.17, LV 0.14
                gK1 = 0.3;// Ik1 conductance
                gnak = 1.5;
                vup = 0.6;//0.4;// uptake strength
                taus = 1.5;//0.75;// diffusional delay (ms)
    // CHANGED
                gna = 12.0;// sodium conductance (mS/micro F)
                taur = 30.0;// spark lifetime (ms)
                taua = 100.0;// NSR-JSR diffusional delay (ms)
                av = 11.3;
                cstar = 90.0;
        #endif
}


void CCell::Prepare(double BCL, int Iter)
{
        if (Iter==0)
        {
                double dciold = 0;
                double dciold2 = 0;
                bool first = false;
                int Tn = BCL*10000/dt, BCLn = BCL/dt, Durn = stimduration/dt;
                for (int tn = 0;tn<Tn;tn++)
                {
                        double t = tn*dt;
```

```
                        if (tn%BCLn < Durn)
                        {
                                if (first)
                                {
                                        if (fabs(ci-dciold2)<0.00001 && t>BCL*300)
                                        {
                                                break;
                                        }
                                        dciold2 = dciold;
                                        dciold = ci;
                                        first = false;
                                }
                                Pace(stim);
                        }
                        else
                        {
                                first = true;
                                Pace();
                        }
                }
        }
        else
        {
                int Tn = BCL*Iter/dt, BCLn = BCL/dt, Durn = stimduration/dt;
                for (int tn = 0;tn<Tn;tn++)
                {
                        if (tn%BCLn < Durn)
                                Pace(stim);
                        else
                                Pace();
                }
        }
}

CCell::~CCell()
{
        delete[] y;
}

CCell& CCell::operator=(const CCell& cell)
{
        if (&cell!=this)
        {
                for (int i=0;i<N;i++)
                {
                        y[i]=cell.y[i];
                }
                vold=cell.vold;
                dt=cell.dt;

                #ifdef ___USE_VAR_FOR_CONST
                        xnao = cell.xnao;
                        xki = cell.xki;
                        xko = cell.xko;
                        cao = cell.cao;

                        gca = cell.gca;
                        gtos = cell.gtos;
                        gtof = cell.gtof;
                        gnaca = cell.gnaca;
                        gkr = cell.gkr;
                        gks = cell.gks;
                        gK1 = cell.gK1;
                        gnak = cell.gnak;
                        vup = cell.vup;
                        taus = cell.taus;
                        gna = cell.gna;
                        taur = cell.taur;
                        taua = cell.taua;
                        av = cell.av;
                        cstar = cell.cstar;
                #endif
        }
```

```
        return(*this);
}

void CCell::ClampAP(double t, double T, double APD)
{
        const double Vmin = -80;//-80mV
        const double Vmax = 30;//30mV
        double clampv;
        if (APD==0)
        {
                const double a=2.0/3.0*1000;
                double x=a/(a+T);
                int m=(int)(t/T);
                if (m*T+x*T>t)
                {
                        clampv = Vmin+(Vmax-Vmin)*sqrt(1-((t-m*T)/x/T)*((t-m*T)/x/T)
);
                }
                else
                {
                        clampv = Vmin;
                }
        }
        else
        {
                double x = APD/T;
                int m = (int)(t/T);
                if (m*T+x*T>t)
                {
                        clampv = Vmin+(Vmax-Vmin)*sqrt(1-((t-m*T)/x/T)*((t-m*T)/x/T)
);
                }
                else
                {
                        clampv = Vmin;
                }
        }

        double dv = (vold-v)/dt;
        vold = v;
        double Itotal;
        if(fabs(dv)>25.0)// then finer time step when dv/dt large
        {
                dtt = dt/10;
                for (int iii=0;iii<10;iii++)
                {
                        v = clampv;
                        Itotal = PaceX(0);
                }
        }
        else
        {
                dtt = dt;
                v = clampv;
                Itotal = PaceX(0);
        }
}

double CCell::Pace(double Istim)
{
        ////////////////////////// time step adjustment ////////////////////////
//
        double dv = (vold-v)/dt;
        vold = v;
        double Itotal;
        //if(fabs(dv)>25.0)// then finer time step when dv/dt large
        if(0)
        {
                dtt = dt/10;
                for (int iii=0;iii<10;iii++)
                {
                        Itotal=PaceX(Istim);
                }
```

```
        }
        else
        {
                dtt = dt;
                Itotal = PaceX(Istim);
        }
        step++;
        return Itotal;
}

double CCell::PaceVClamp(double clampv)
{
        ////////////////////////// time step adjustment ////////////////////////
//
        double dv = (vold-v)/dt;
        vold = v;
        double Itotal;
        if(fabs(dv)>25.0)// then finer time step when dv/dt large
        {
                dtt = dt/10;
                for (int iii=0;iii<10;iii++)
                {
                        v = clampv;
                        Itotal = PaceX(0);
                        v = clampv;
                }
        }
        else
        {
                dtt = dt;
                v = clampv;
                Itotal = PaceX(0);
                v = clampv;
        }
        return Itotal;
}

double CCell::PaceX(double Istim)
{
        double xik1 = comp_ik1();
        double xito = comp_ito();//itos and itof
        double xinak = comp_inak();
        double csm = cs/1000.0;// convert micro M to mM
        double Jnaca = comp_inaca(csm);

        ////////////////////// Equations for Ca cycling ////////////////////
        double xdif = (cs-ci)/taus;//diffusion from submembrane to myoplasm
        // Troponin kinetics
        const double xkon = 0.0327;
        const double xkoff = 0.0196;
        const double btrop = 70.0;
        double xbi = xkon*ci*(btrop-tropi)-xkoff*tropi;
        double xbs = xkon*cs*(btrop-trops)-xkoff*trops;

        double xiup = comp_iuptake();
        double xileak = comp_ileak();

        ica = comp_ica(csm);
        double JCa = comp_hh_ical(ica,csm);
        double xsvipca = comp_svipca();// PMCA

        double dcs = comp_inst_buffer(cs)*(50.0*(xir-xdif-JCa+Jnaca-xsvipca/16.0)-xb
s); //----with PMCA----
        double dci = comp_inst_buffer(ci)*(xdif-xiup+xileak-xbi);
        double dcj = -xir+xiup-xileak;// SR load dynamics
        double dcjp = (cj-cjp)/taua;// NSR-JSR relaxation dynamics
        double Qr = comp_Q();
        double dir = comp_dir(Qr,JCa,dcj);

        double xina = comp_ina();
        double xikr = comp_ikr();
        double xiks = comp_iks();
```

```
        cs += dcs*dtt;
        ci += dci*dtt;
        cj += dcj*dtt;
        cjp += dcjp*dtt;
        xir += dir*dtt;

        tropi += xbi*dtt;
        trops += xbs*dtt;

        ////////////////////// convert ion flow to current ////////////////////
        const double wca = 8.0;//conversion factor between micro molar/ms to micro a
mps/ micro farads
        double xinaca = wca*Jnaca;
        double xical = 2.0*wca*JCa;
        jrel = xical;

        ////////////////////// sodium dynamics ////////////////////
        const double xrr = (1.0/wca)/1000.0;// note: sodium is in m molar so need to
 divide by 1000
        //xnai += 100*(-xrr*(xina+3.0*xinak+3.0*xinaca))*dtt;

        ////////////////////// dV/dt ////////////////////////////////
        double Itotal = ( -( xina + xik1 + xikr + xiks + xito + xinaca + xical + xin
ak /*+xsvipca*/ ) + Istim); //with PMCA
        v += Itotal*dtt;

        ////////////////////// output to file ////////////////////
        _inaca=xinaca;  _ical=xical;    _iks=xiks;      _ikr=xikr;
    _ik1=xik1;
        _ina=xina;        _inak=xinak;          _iup=xiup;      _svipca=xsvipca;
    _up=xiup;
        _ir=xir;

        return Itotal;
}

////////////////////////////// PMCA ////////////////////////////
double CCell::comp_svipca(void)
{
        double xsvipca = 0.18*1.15*cs/(3.6+cs);
        return xsvipca;
}

////////////////////////////// sodium current following Hund-Rudy ////////////////////
////////
double CCell::comp_ina(void)
{
        double ena = (1.0/frt)*log(xnao/xnai);
        double am;
        double vdna = 0;
        if (fabs(v+47.13+vdna)<0.001/0.1)
                am=3.2;
        else
                am = 0.32*(v+47.13+vdna)/(1.0-exp(-0.1*(v+47.13+vdna)));
        double bm = 0.08*exp(-(v+16+vdna)/11.0);
        double ah,bh,aj,bj;
        if(v<(-40.0))
        {
                 ah = 0.135*exp((80.0+v)/(-6.8));
                 bh = 3.56*exp(0.079*v)+310000.0*exp(0.35*v);
                 aj = ((-127140.0*exp(0.2444*v)-0.00003474*exp(-0.04391*v))*(v+37.78
))/(1.0+exp(0.311*(v+79.23)));//*0.5;
                 bj = (0.1212*exp(-0.01052*v))/(1.0+exp(-0.1378*(v+40.14)));//*0.5;
        }
        else
        {
                 ah = 0.0;
                 bh = 1.0/(0.13*(1.0+exp((v+10.66)/(-11.1))));
                 aj = 0.0;
                 bj = (0.3*exp(-0.0000002535*v))/(1.0+exp(-0.1*(v+32.0)));//*0.5;
        }

        double tauh = 1.0/(ah+bh);
```

```
        double tauj = 1.0/(aj+bj);
        double taum = 1.0/(am+bm);
        double xina = gna*xh*xj*xm*xm*xm*(v-ena);

        xh = ah/(ah+bh)-((ah/(ah+bh))-xh)*exp(-dtt/tauh);
        xj = aj/(aj+bj)-((aj/(aj+bj))-xj)*exp(-dtt/tauj);
        xm = am/(am+bm)-((am/(am+bm))-xm)*exp(-dtt/taum);

        return xina;
}

//////////////////// MGWMN model, new fitted ///////////////////////

double CCell::comp_ikr(void)
{
        double alpha1 = 0.007953 * exp(v/30.631250);
        double beta1 = 0.083063 * exp(-v/69.176);
        double alpha2 = 0.008511 * exp(v/21.167);
        double beta2 = 0.003242 * exp(v/95.020218);
        double alphai = 0.187367 * exp(v/1467.569512);
        double betai = 0.044965 * exp(-v/25.197271);
        double alphai2 = 0.000035 * exp(v/496052.439941);
        double sai = alphai2*betai*beta2/alpha2/alphai;
        double kf = 0.300789;
        double kb = 0.135702;

        double dc1dt = - IKrC1 * alpha1 + IKrC2 * beta1;
        double dc2dt =   IKrC1 * alpha1 - IKrC2 * beta1 + IKrC3 * kb - IKrC2 * kf;
        double dc3dt = - IKrC3 * kb + IKrC2 * kf - IKrC3 * alpha2 + IKrO * beta2 - I
KrC3 * alphai2 + IKrI * sai;
        double dodt =    IKrC3 * alpha2 - IKrO * beta2 + IKrI * betai - IKrO * alpha
i;
        double didt =  - IKrI * betai + IKrO * alphai - IKrI * sai + IKrC3 * alphai2
;

        IKrC1 += dc1dt * dtt;
        IKrC2 += dc2dt * dtt;
        IKrC3 += dc3dt * dtt;
        IKrO += dodt * dtt;
        IKrI += didt * dtt;

        double xikr = gkr*sqrt(xko/5.4)*IKrO*(v-ek);

        return xikr;
}

//////////////////// HH model, new fitted ///////////////////////
/*
double CCell::comp_ikr(void)
{
        const double gss=sqrt(xko/5.4);
        double xkrinf = 1/(1+exp(-(v+7.78)/7.14));
        double ykrinf = 1/(1+exp((v+53.5)/24.06));
        double xkrtau = 1/(1.466*(v-214.2)/(1-exp(-(v-214.2)/17.18))+0.01751*(v+58.9
3)/(exp((v+58.93)/3.779)-1));

        double xikr=gkr*gss*IKrO*IKrI*(v-ek);

        IKrO = xkrinf-(xkrinf-IKrO)*exp(-dtt/xkrtau);
        IKrI = ykrinf; // instantaneous

        return xikr;
}
*/

//////////////////////////// Iks modified from Shannon, with new Ca dependence /////
///////////////
double CCell::comp_iks(void)
{
        const double prnak = 0.018330;
        double eks = (1.0/frt)*log((xko+prnak*xnao)/(xki+prnak*xnai));
        double xs1ss = 1.0/(1.0+exp(-(v-1.50)/16.70));
        double xs2ss = xs1ss; // modified
```

```
        double tauxs1;
        if (fabs(v+30.0)<0.001/0.0687)
                tauxs1 = 1/(0.0000719/0.148+0.000131/0.0687);
        else
                tauxs1 = 1.0/(0.0000719*(v+30.0)/(1.0-exp(-0.148*(v+30.0)))+0.000131
*(v+30.0)/(exp(0.0687*(v+30.0))-1.0));
        double tauxs2 = 1.0*tauxs1;
        double gksx = (1+0.8/(1+pow((0.3/ci),3)));
        double xiks = gks*gksx*xs1*xs2*(v-eks);
        xs1 = xs1ss-(xs1ss-xs1)*exp(double(-dtt/tauxs1));
        xs2 = xs2ss-(xs2ss-xs2)*exp(double(-dtt/tauxs2));
        return xiks;
}

////////////////////////// Ik1 following Luo-Rudy formulation (from Shannon model)
 ////////////////////////
double CCell::comp_ik1(void)
{
        const double gki = (sqrt(xko/5.4));
        double aki = 1.02/(1.0+exp(0.2385*(v-ek-59.215)));
        double bki = (0.49124*exp(0.08032*(v-ek+5.476))+exp(0.061750*(v-ek-594.31)))
/(1.0+exp(-0.5143*(v-ek+4.753)));
        double xkin = aki/(aki+bki);
        double xik1 = gK1*gki*xkin*(v-ek);
        return xik1;
}

////////////////////////// Ito slow following Shannon et. al. 2005 //////////////
/////////////
double CCell::comp_ito(void)
{
        double xtos_inf = 1/(1+exp(-(v+10.06)/13.75));
        double ytos_inf = 1.0/(1.0+exp((v+41.97)/5.14));
        double txs = 9.0/(1.0+exp((v+3)/15.0)) + 0.5;
        double tys =  80 + 3000/(1+exp((v+60.)/10.)); // RV: 80, LV: 50
        double xitos = gtos*xtos*(ytos)*(v-ek);// ito slow
        xtos = xtos_inf-(xtos_inf-xtos)*exp(-dtt/txs);
        ytos = ytos_inf-(ytos_inf-ytos)*exp(-dtt/tys);

        double xtof_inf = xtos_inf;
        double ytof_inf = ytos_inf;
        double txf = 9.0/(1.0+exp((v+3)/15.0)) + 0.5;
        double tyf = 8.4 + 100/(1+exp((v+60.)/10.));
        double xitof = gtof*xtof*ytof*(v-ek); // ito fast
        xtof = xtof_inf-(xtof_inf-xtof)*exp(-dtt/txf);
        ytof = ytof_inf-(ytof_inf-ytof)*exp(-dtt/tyf);

        // output to file
        _itof=xitof; _itos=xitos;

        return xitos+xitof;
}

////////////////////////// Inak (sodium-potassium exchanger) following Shannon ///
//////////////////////
double CCell::comp_inak(void)
{
        const double xkmko = 1.5;       //these are Inak constants adjusted to fit
                                                //the experimentally measure
d dynamic restitution curve
        const double xkmnai = 12.0;
        const double sigma = (exp(xnao/67.3)-1.0)/7.0;
        double fnak = 1.0/(1+0.1245*exp(-0.1*v*frt)+0.0365*sigma*exp(-v*frt));
        double xinak = gnak*fnak*(1./(1.+(xkmnai/xnai)))*xko/(xko+xkmko);
        return xinak;
}

////////////////////////// Inaca (sodium-calcium exchange) following Shannon and H
und-Rudy ////////////////////////
//      Note: all concentrations are in mM
double CCell::comp_inaca(double csm)
{
        double zw3 = pow(xnai,3)*cao*exp(v*0.35*frt)-pow(xnao,3)*csm*exp(v*(0.35-1.)
```

```
*frt);
        double zw4 = 1.0+0.2*exp(v*(0.35-1.0)*frt);
        const double xkdna = 0.3;// micro M
        double aloss = 1.0/(1.0+pow((xkdna/cs),3));
        const double xmcao = 1.3;
        const double xmnao = 87.5;
        const double xmnai = 12.3;
        const double xmcai = 0.0036;
        double yz1 = xmcao*pow(xnai,3)+pow(xmnao,3)*csm;
        double yz2 = pow(xmnai,3)*cao*(1.0+csm/xmcai);
        double yz3 = xmcai*pow(xnao,3)*(1.0+pow((xnai/xmnai),3));
        double yz4 = pow(xnai,3)*cao+pow(xnao,3)*csm;
        double zw8 = yz1+yz2+yz3+yz4;
        double Jnaca = gnaca*aloss*zw3/(zw4*zw8);
        return Jnaca;
}

////////////////////////// compute driving force: i_Ca //////////////////////////
/
double CCell::comp_ica(double csm)
{
        const double pca = 0.00054;
        double za = v*2.0*frt;
        double factor1 = 4.0*pca*xf*xf/(xxr*temp);
        double factor = v*factor1;
        double ica;
        if(fabs(za)<0.001)
                ica = factor1*(csm*exp(za)-0.341*(cao))/(2.0*frt);
        else
                ica = factor*(csm*exp(za)-0.341*(cao))/(exp(za)-1.0);

        return ica;
}


////////////////////// HH Ca current ////////////////////////////
double CCell::comp_hh_ical(double ica,double csm)
{
        double dinf = 1/pow2(1+exp(-(v+17.6)/8.1));
        double taud = 0.8;
        double finf = 0.98/(1+exp((v+27.66)/4.0432))+0.5/(1+exp(-(v-41)/8.93))+0.02;
 // MGWMN IKr
        double tauf = 75.0/(1 + exp(-(v - 30.0)/4.0)) + 25.0; // MGWMN IKr
        //double finf = 0.98/(1+exp((v+27.66)/4.0432))+0.5/(1+exp(-(v-46)/8.93))+0.0
2; // HH IKr
        //double tauf = 43.0/(1 + exp(-(v - 30.0)/6.0)) + 25.0; // HH IKr
        double finf_ca = 0.7*exp(-csm*1000/0.42)+0.3*exp(-csm*1000/9);
        double tauf_ca = 3;
        double JCa = gca*hd*hf*hf_ca*ica;

        hd = dinf-(dinf-hd)*exp(double(-dtt/taud));
        hf = finf-(finf-hf)*exp(double(-dtt/tauf));
        hf_ca = finf_ca-(finf_ca-hf_ca)*exp(double(-dtt/tauf_ca));

        return JCa;
}

////////////////////// SERCA2a uptake current ////////////////////////////////////
////////
double CCell::comp_iuptake(void)
{
        const double xup = 0.5;// uptake threshold
        double xiup = vup*ci*ci/(ci*ci + xup*xup);              //----CHANGED
        return xiup;
}

////////////////////////// leak from the SR ////////////////////////////
double CCell::comp_ileak(void)
{
        const double gleak = 0.00002069;
        return gleak*(cj*cj/(cj*cj+50.0*50.0))*(cj*16.667-ci);
}
```

```
//////////////////////////// buffer dynamics in the myoplasm //////////////////////
/////
//buffering to calmodulin and SR are instantaneous, while buffering to
//Troponin C is time dependent.These are important to have reasonable
//Ca transient. Note: we have buffering in the submembrane space and
//the myoplasm.
double CCell::comp_inst_buffer(double c)
{
        const double bcal = 24.0;
        const double xkcal = 7.0;
        const double srmax = 47.0;
        const double srkd = 0.6;
        const double bmem = 15.0;
        const double kmem = 0.3;
        const double bsar = 42.0;
        const double ksar = 13.0;
        double bpx = bcal*xkcal/((xkcal+c)*(xkcal+c));
        double spx = srmax*srkd/((srkd+c)*(srkd+c));
        double mempx = bmem*kmem/((kmem+c)*(kmem+c));
        double sarpx = bsar*ksar/((ksar+c)*(ksar+c));
        return 1.0/(1.0+bpx+spx+mempx+sarpx);
}

//////////////////////////// release-load functional dependence ////////////////////
////////
double CCell::comp_Q(void)
{
        double bv = (cstar-50.)-av*cstar;
        double Qr;
        if (cjp<50)
                Qr = 0.0;
        else if (cjp>50.0 && cjp<cstar)
                Qr = cjp-50.0;
        else
                Qr = av*cjp+bv;

        return cj*Qr/cstar;
}

double CCell::comp_dir(double Qr, double JCa, double dcj)
{
        double sparkV = 1/(1.+exp((v+25)/25))/(1+exp(-(v+37)/5));
        double spark_rate = 1.29*fabs(JCa)/gca*sparkV*fspark;
        double alpha = 80.0;
        double dfdt = -alpha*spark_rate + (1-fspark)/500.0;

        fspark += dfdt*dtt;

        return spark_rate*Qr-xir*(1-taur*dcj/cj)/taur;
}
```