Computational Electronics: Numerical Details

Dragica Vasileska and Gerhard Klimeck

1. To obtain diagonally-dominant coefficient matrix when using finite difference scheme for the discretization of the Poisson equation, it is necessary to use some linearization scheme. The simplest way to achieve this is to use $\psi \to \psi + \delta$, where δ is small.
   (a) Write down the linearized Poisson equation using this linearization scheme.
   (b) Write down (derive) the scaled version of the result obtained in (a).
   (c) Write the finite-difference approximation for the scaled Poisson equation.
   (d) If one solves (c) for the improvement δ, show that the resultant coefficient matrix **A** is diagonally dominant.

   (Note: Matrix **A** is diagonally dominant if the absolute value of the sum of the off-diagonal elements in each row is smaller than the absolute value of the corresponding diagonal term.)

2. Develop a one-dimensional (1D) drift-diffusion simulator for modeling *pn*-junctions (diodes) under forward and reverse bias conditions. Include both types of carriers in your model (electrons and holes). Use the finite-difference expressions for the electron and hole current continuity equations using Sharfetter-Gummel discretization scheme, which was described in the class.

   Model: Silicon diode, with permittivity $\varepsilon_{sc} = 1.05 \times 10^{-10}$ F/m and intrinsic carrier concentration $n_i = 1.5 \times 10^{10}$ cm^{-3} at T=300K. In all your simulations assume that T=300K. Use concentration-dependent and field-dependent mobility models and SRH generation-recombination process. Assume ohmic contacts and charge neutrality at both ends to get the appropriate boundary conditions for the potential and the electron and hole concentrations.

   - For the electron and hole mobility use 1500 and 1000 cm^2/V-s, respectively.

- For the SRH generation-recombination, use TAUN0=TAUP0=0.1 us. To simplify your calculations, assume that the trap energy level coincides with the intrinsic level.

Doping: Use $N_A = 10^{16} cm^{-3}$ and $N_D = 10^{17} cm^{-3}$ as a net doping of the *p*- and *n*-regions, respectively.

Numerical methods: Use the LU decomposition method for the solution of the 1D Poisson and the two 1D continuity equations for electrons and holes individually. Use Gummel's decoupled scheme, described in the class, to solve the resultant set of coupled set of algebraic equations.

Outputs:

- Plot the conduction band edge under equilibrium conditions (no current flow) and for VA=0.625 V.
- Plot the electron and hole densities under equilibrium conditions (no current flow) and for VA=0.625 V.
- Vary the Anode bias V_A from 0 to 0.625 V, in voltage increments that are fraction of the thermal voltage $V_T = k_B T / q$, to have stable convergence. Plot the resulting I-V characteristics. The current will be in A/unit area, since you are doing 1D modeling. Check the conservation of current when going from the cathode to the anode, which also means conservation of particles in your system. For the calculation of the current density, use the results given in the notes.
- For V_A =0.625 V, plot the position of the electron and hole quasi-Fermi levels, with respect to the equilibrium Fermi level, assumed to be the reference energy level.

Final note: When you submit your project report, in addition to the final results, give a brief explanation of the problem you are solving with reference to the listing of your program that you need to turn in with the report.

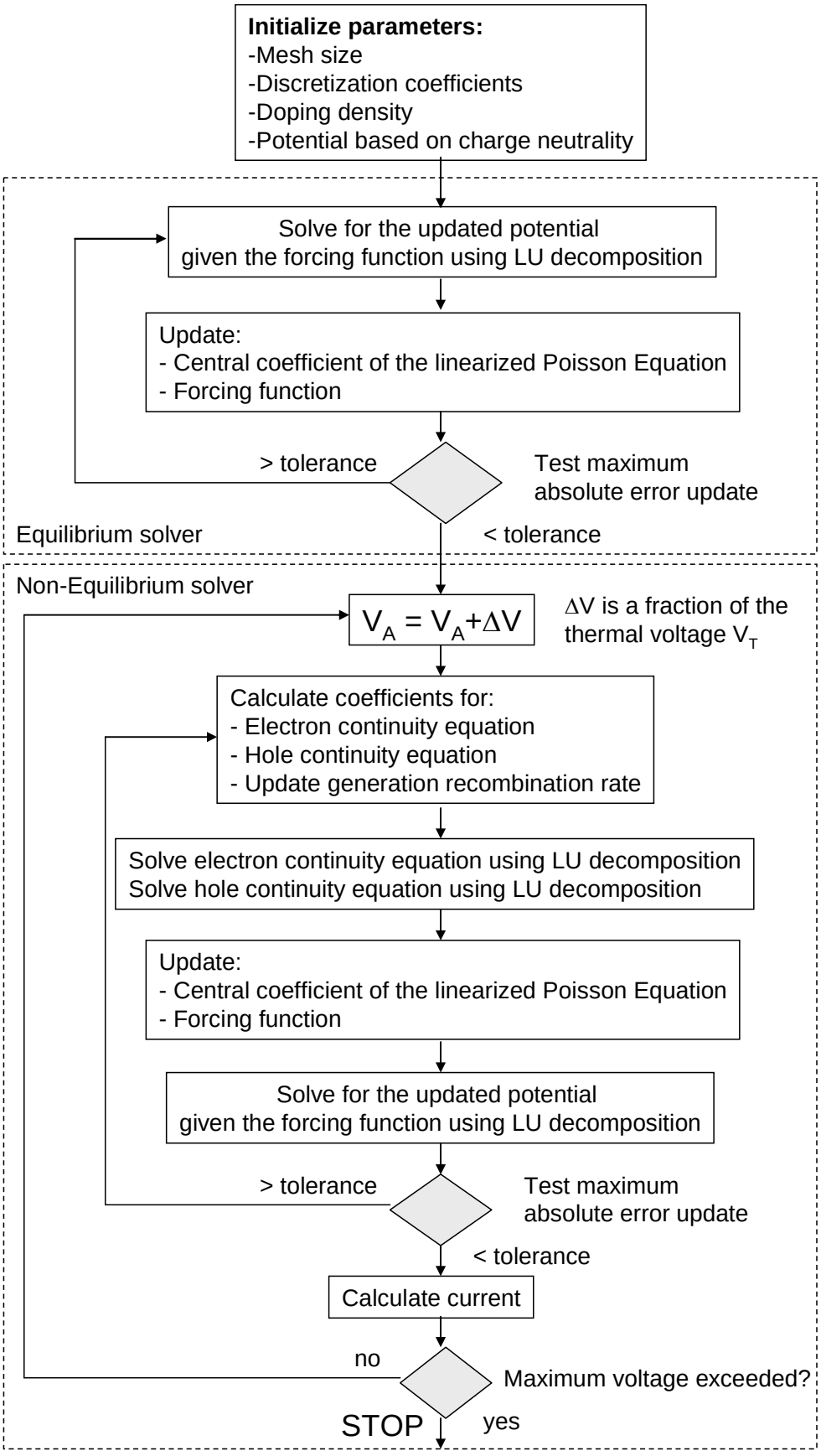
```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    CCCCCCCC
C
C      1D Drift-Diffusion simulator for pn diodes
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    CCCCCCCC

      implicit real*8(a-h, o-z)
      parameter(max_val = 40000)
      real*8 kb, ni, Na, Nd, Ldn, Ldp, Ldi
```

```
      real*8 Ncn, Ncp, NN, Nsrh_n, Nsrh_p
      real*8 dop(0:max_val+1),fi(0:max_val+1), delta(0:max_val+1)
      real*8 a(0:max_val+1),b(0:max_val+1),c(0:max_val+1)
        real*8 f(0:max_val+1)
        real*8 d(0:max_val+1),v(0:max_val+1)
      real*8 n(0:max_val+1),p(0:max_val+1)
      logical flag_conv
      integer method

C.....Define fundamental constants and material parameters:
      q   = 1.602D-19
      kb  = 1.38D-23
      eps = 1.05D-12
      T   = 300
      ni  = 1.5D10
      Vt  = kb*T/q
      RNc = 2.8D20
      dEc = Vt*dlog(Rnc/ni)

C.....Read doping:
      Print*,'Acceptor doping concentration:'
      read*,Na
      Print*,'Donor doping concentration:'
      read*,Nd
      print*,'Maximum applied bias'
      read*,Va_max
      print*,'Voltage step'
      read*,dVa

C.....Calculate relevant parameters for the simulation:
C       (1) Built-in voltage:
       Vbi = Vt*dlog(Na/ni*Nd/ni)
       W   = dsqrt(2.*eps*(Na+Nd)*Vbi/q/Na/Nd)
       Wn  = W*Na/(Na+Nd)
       Wp  = W*Nd/(Na+Nd)
       E_p = q*Nd*Wn/eps
       Ldn = dsqrt(eps*Vt/q/Nd)
       Ldp = dsqrt(eps*Vt/q/Na)
       Ldi = dsqrt(eps*Vt/q/ni)

       open(unit=1,file='input.params',status='unknown')
       write(1,*)'Na[cm-3]=',Na
       write(1,*)'Nd[cm-3]=',Nd
       write(1,*)'Vbi[V]=',Vbi
       write(1,*)'W[cm]=',W
       write(1,*)'Wn[cm]=',Wn
       write(1,*)'Wp[cm]=',Wp
       write(1,*)'E_peak[V/cm]=',E_p
       write(1,*)'Ldn[cm]=',Ldn
       write(1,*)'Ldp[cm]=',Ldp
       write(1,*)'  '
       write(1,*)'  '
       write(1,*)'Convergence of the outer loop'

C.....Define some material constants:
       Ncn = 1.432D17
       rmu_1n = 88.D0
       rmu_2n = 1252.D0
       Ncp = 2.67D17
       rmu_1p = 54.3D0
       rmu_2p = 407.D0
       tau_n0 = 1.D-7
       tau_p0 = 1.D-7
       Nsrh_n = 5.D16
       Nsrh_p = 5.D16

C.....Setting the size of the simulation domain based
C       on the analytical results for the width of the depletion regions
       x_max = 0.
       if(x_max.lt.Wn)x_max = Wn
```

```
      if(x_max.lt.Wp)x_max = Wp
      x_max = 70*x_max

C.........Setting the grid size based on the extrinsic Debye lengths:
      dx = Ldn
      if(dx.gt.Ldp)dx=Ldp
      dx = dx/20

C.........Calculate the required number of grid points and renormalize dx:
      n_max = x_max/dx
      print*,n_max
      if(n_max.gt.max_val)then
        print*,'Maximum array size exceeded!'
        goto 111
      endif
      dx = dx/Ldi

C.........Set up the doping C(x)=Nd(x)-Na(x)that is normalized with ni:
      do i = 0,n_max+1
        if(i.le.n_max/2)then
          dop(i) = -Na/ni
        else
          dop(i) = Nd/ni
        endif
      enddo

C.........Initialize the potential based on the requirement of charge
C         neutrality throughout the whole structure:
      do i = 0, n_max+1
        zz = 0.5*dop(i)
        if(zz.gt.0)then
          xx = zz*(1.+dsqrt(1.+1./(zz*zz)))
        elseif(zz.lt.0)then
          xx = zz*(1.-dsqrt(1.+1./(zz*zz)))
        endif
        fi(i) = dlog(xx)
      enddo

      delta_acc = 1.D-6                 ! preset tolerance

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
     CCCCCCCC
C
C        SOLVE THE EQUILIBRIUM CASE:
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
     CCCCCCCC

C    (A) Define the elements of the coefficient matrix for the internal nodes and
C        initialize the forcing function:
      dx2 = dx*dx
      do i = 2, n_max-1
        a(i) = 1./dx2
        c(i) = 1./dx2
        b(i) = -(2./dx2+dexp(fi(i))+dexp(-fi(i)))
        f(i) = dexp(fi(i))-dexp(-fi(i))-dop(i) -
   1          fi(i)*(dexp(fi(i))+dexp(-fi(i)))
      enddo

C    (B) Define the elements of the coefficient matrix and initialize the forcing
C        function at the ohmic contacts
      a(1) = 0.
      c(1) = 0.
      b(1) = 1.
      f(1) = fi(1)
      a(n_max) = 0.
      c(n_max) = 0.
      b(n_max) = 1.
      f(n_max) = fi(n_max)
```

```
C    (C)  Start the iterative procedure for the solution of the linearized Poisson
C         equation using LU decomposition method:

         flag_conv = .false.                    ! convergence of the Poisson loop
         k _iter= 0
         do while(.not.flag_conv)
           k_iter = k_iter + 1
           d(1) = b(1)
           do i = 2, n_max
             d(i) = b(i) - a(i)*c(i-1)/d(i-1)
           enddo
C            Solution of Lv = f:
           v(1) = f(1)
           do i = 2, n_max
             v(i) = f(i) - a(i)*v(i-1)/d(i-1)
           enddo
C            Solution of U*fi=v:
           temp = v(n_max)/d(n_max)
           delta(n_max) = temp - fi(n_max)
           fi(n_max)=temp
           do i = n_max-1,1,-1
             temp = (v(i)-c(i)*fi(i+1))/d(i)
             delta(i) = temp - fi(i)
             fi(i) = temp
           enddo

C            Test update in the outer iteration loop:
           delta_max = 0.
           do i = 1, n_max
             xx = dabs(delta(i))
             if(xx.gt.delta_max)delta_max=xx
           enddo

           print*,k_iter, delta_max
           write(1,*)k_iter, delta_max

C            Test convergence and recalculate forcing function and
C            central coefficient b if necessary:
           if(delta_max.lt.delta_acc)then
             flag_conv = .true.
           else
             do i = 2, n_max-1
               b(i) = -(2./dx2+dexp(fi(i))+dexp(-fi(i)))
               f(i) = dexp(fi(i))-dexp(-fi(i))-dop(i)-
     1               fi(i)*(dexp(fi(i))+dexp(-fi(i)))
             enddo
           endif

         enddo

C    (D)  Write the results of the simulation in files:
         open(unit=3,file='cond_band',status='unknown')
         open(unit=4,file='tot_charge',status='unknown')
         open(unit=5,file='el_field',status='unknown')
         open(unit=6,file='np_data',status='unknown')
         open(unit=7,file='quasi_ef',status='unknown')
         open(unit=8,file='curr_profile',status='unknown')
         open(unit=9,file='Ia_vs_Va',status='unknown')
         xx = 0.
         do i = 1, n_max
           write(3,*)xx,dEc-Vt*fi(i)
           ro = -q*ni*(dexp(fi(i))-dexp(-fi(i))-dop(i))
           write(4,*)xx,ro
           if(i.gt.1)then
             el_field1 = -(fi(i+1)-fi(i))*Vt/dx/Ldi
             el_field2 = -(fi(i+1)-fi(i-1))*Vt/(2.*dx*Ldi)
             write(5,*)xx,el_field1,el_field2
           endif
           n(i) = dexp(fi(i))
           p(i) = dexp(-fi(i))
```

```
          write(6,*)xx,n(i)*ni,p(i)*ni
          xx = xx + dx*Ldi
        enddo
        close(3)
        close(4)
        close(5)
        close(6)
        close(7)
        close(8)
        close(9)

111   continue

      end


      FUNCTION BER(X)

      IMPLICIT REAL*8(A-H, O-Z)
      LOGICAL FLAG_SUM

      FLAG_SUM = .FALSE.
        if(x.gt.0.01)then
          Ber = x*dexp(-x)/(1.-dexp(-x))
        elseif(x.lt.0.and.dabs(x).gt.0.01)then
          Ber = x/(dexp(x)-1.)
        elseif(x.eq.0)then
          Ber = 1.D0
        else
          temp_term = 1.
          sum = temp_term
          i = 0.
          do while(.not.flag_sum)
            i = i + 1
            temp_term = temp_term*x/dfloat(i+1)
            if(sum+temp_term.eq.sum)flag_sum = .true.
            sum = sum + temp_term
          enddo
          Ber = 1./sum
        endif
      RETURN
      END
```


Initialize parameters:
-Mesh size
-Discretization coefficients
-Doping density
-Potential based on charge neutrality
Solve for the updated potential
given the forcing function using LU decomposition
Update:
- Central coefficient of the linearized Poisson Equation
- Forcing function
> tolerance
Test maximum
absolute error update
Equilibrium solver
< tolerance
Non-Equilibrium solver
$V_A = V_A + \Delta V$
ΔV is a fraction of the
thermal voltage V_T
Calculate coefficients for:
- Electron continuity equation
- Hole continuity equation
- Update generation recombination rate
Solve electron continuity equation using LU decomposition
Solve hole continuity equation using LU decomposition
Update:
- Central coefficient of the linearized Poisson Equation
- Forcing function
Solve for the updated potential
given the forcing function using LU decomposition
> tolerance
Test maximum
absolute error update
< tolerance
Calculate current
no
Maximum voltage exceeded?
STOP
yes