rrvUEDGE equations, from Version 4.33 manual

**Poloidal ion velocity:**



**Radial ion velocity:**



**Electron velocities:**



**Ion continuity equation:**



**Ion parallel momentum equation (momentum due to E?):**



**Electron energy equation:**



**Ion Energy equation:**



**Potential equations:**



**Neutral gas density (diffusive neutrals):**



**Neutral parallel momentum equation (reduced Navier-Stokes):**



**Scalars**

|  |  |
| --- | --- |
| nisp | Number of plasma species (2 = neutral D) |
| ngsp | Number of gas species |
| mg | Gas species mass: [kg] |
| minu | Ion mass in units of proton mass: [AMU] |
| zi | Atom charge |
| zn | Nuclear charge |
| qe or ev | Electron charge: 1.6e-19 |
| mn(0:nx+1, 0:ny, nisp) | Mass density |
|  |  |

**State variables**

|  |  |
| --- | --- |
| ni(0:nx+1, 0:ny+1,1:nisp) | Ion density at cell center: (poloidal, radial, species, nisp=2: neutral deuterium) |
| niy0(0:nx+1, 0:ny+1, 1:ngsp) | Ion density below y-face center |
| niy1(0:nx+1, 0:ny+1, 1:ngsp) | Ion density above y-face center |
| ne(0:nx+1, 0:ny+1) | Electron density |
| ng(0:nx+1, 0:ny+1, 1:ngsp) | Neutral density |
| te(0:nx+1, 0:ny+1) | Electron temperature [J] |
| ti(0:nx+1, 0:ny+1) | Ion temperature [J]: same for all ions |
| pr(0:nx+1, 0:ny+1) | Total pressure [J/m3], hydrogen + electron, at cell center |
| pre(0:nx+1, 0:ny+1) | Eletron pressure at cell center |
| pri(0:nx+1, 0:ny+1, 1:nisp) | Ion pressure at cell center |
|  |  |
|  |  |
| up(0:nx+1, 0:ny+1,1:nisp) | Parallel velocity at mass-density centre |
| uu(0:nx+1, 0:ny+1, 1:nisp) | Total ion poloidal velocity at x-face = ratio poloidal flux over ion density:  fnix(0:nx+1, 0:ny+1, 1:nisp)/ sx(0:nx+1,0:ny+1)/ni(0:nx+1, 0:ny+1, 1:nisp) |
| uup(0:nx+1, 0:ny+1, 1:nisp) (= rrv(0:nx+1, 0:ny+1) \* up(0:nx+1, 0:ny+1, 1:nisp) | Poloidal ion velocity due to Bpol/B x up |
| rbfbt(0:nx+1, 0:ny+1) \* v2ce(0:nx+1, 0:ny+1, 1:nisp) | Poloidal ion velocity due ExB |
| rbfbt(0:nx+1, 0:ny+1) \* v2cb(0:nx+1, 0:ny+1, 1:nisp) | Poloidal ion velocity due to Grad(B) |
| rbfbt(0:nx+1, 0:ny+1) \* v2rd(0:nx+1, 0:ny+1, 1:nisp) | Poloidal ion velocity due to resistive drift |
| rbfbt(0:nx+1, 0:ny+1) \* v2dd(0:nx+1, 0:ny+1, 1:nisp) | Poloidal ion velocity due to diagmagnetic drift |
|  |  |
| -uup(,,isp)/rrv\*rbfbt+sign(1.,b0)\*((1-cftef)\*v2ce(,,isp)+(1-cftdd)\*v2cd(,,isp))\*rrv | Toroidal ion velocity |
|  |  |
| vy(0:nx+1, 0:ny+1, 1:nisp) | Radial ion velocity at y-face |
| vyce(0:nx+1, 0:ny+1, 1:nisp) | Radial ion velocity due to ExB: v(iy,E) |
| vycb(0:nx+1, 0:ny+1, 1:nisp) | Radial ion velocity due to Grad B: v(iy, gradB) |
| vydd(0:nx+1, 0:ny+1, 1:nisp) | Radial ion velocity due to anomalous drift:  -Danom/ni dni/dy |
| vyrd(0:nx+1, 0:ny+1, 1:nisp) | Radial ion velocity due to resistive drift |
|  |  |
| upe(0:nx+1, 0:ny+1) | Parallel electron velocity |
| vex(0:nx+1, 0:ny+1) | Poloidal electron velocity |
| vey(0:nx+1, 0:ny+1) | Radial electron velocity |
|  |  |
| veycb(0:nx+1,0:ny+1) | Radial electron velocity from grad\_B |
| veycp(0:nx+1, 0:ny+1) | Radial electron velocity from grad\_PeXB |
| rbfbt(0:nx+1, 0:ny+1)\*ve2cb(0:nx+1,0:ny+1) | Poloidal electron velocity from v2 due to grad\_B |
|  |  |
|  |  |
| phi(0:nx+1, 0:ny+1) | Electric potential: [V] |
|  |  |
| zeff(0:nx+1, 0:ny+1) | Zeff in primary cell |
|  |  |
|  |  |
|  |  |
|  |  |

**Particle and heat sources/sinks**

|  |  |
| --- | --- |
| nuiz(0:nx+1,0:ny+1, ngsp) | Ionization rate: ne\*sigma\*v: [1/s] |
| nucx(0:nx+1,0:ny+1, ngsp) | Charge-exchange rate for neutrals: ni\*sigvi |
| nucxi(0:nx+1,0:ny+1, nisp) | Charge-exchange rate for ions: ng\*sigvg |
| nurc(0:nx+1, 0:ny+1, ngsp) | Recombination rate for neutrals: ng\*sigvi |
| psor(0:nx+1, 0:ny+1, 1:nisp) | Plasma ionization source (> 0): <sigma\_ioniz\*ve> [part/s] |
| psorxr(0:nx+1, 0:ny+1, 1:nisp) | Plasma recombination source (< 0): <sigma\_recomb\*ve> [part/s] |
| psorg(0:nx+1, 0:ny+1, 1:ngsp) | Neutral ionization source: <sigma\_ioniz\*ve> [part/s] |
| psorrg(0:nx+1, 0:ny+1, 1:ngsp) | Neutral recombination source: <sigma\_recomb\*ve> [part/s] |
| psordis(0:nx+1, 0:ny+1) | Dissociation source for hydrogen |
|  |  |
| eeli(0:nx+1, 0:ny+1) | Electron energy loss per ionization [J] |
| vsoree(0:nx+1,0:ny+1) | Combined electron energy source from ionization and recombination with H [J/s] |
| erliz(0:nx+1, 0:ny+1, | H radiation loss for ionization [J] |
| erlrc(0:nx+1, 0:ny+1) | H radiation loss for recombination [W] |
| cnsor\*erlrc(0:nx+1, 0:ny+1) | H radiation from recombination; cnsor = coefficient for particle source in density equation [W] |
| psor(0:nx+1, 0:ny+1)\*(eeli(,)-ebind\*ev)) | H radiation from ionization; eeli = electron energy loss per ionization, ebind = binding energy (13.6 eV) |
| ebind\*ev\*psor(0:nx+1, 0:ny+1) | Binding energy, carried by ion upon ionization |
| peirad(0:nx+1, 0:ny+1) | Total power lost by electrons **and** ions in H radiation, ionization, and H2 dissociation |
| pradht | Domain-integrated H photon radiation loss (-binding energy): [W] |
| pradiz | Contribution of ioniziation only to pradht |
| pradrc | Contribution of recombination only to pradht |
|  |  |
| prad(0:nx+1, 0:ny+1) | Total impurity radiation [W/m3] |
| pradz(0:nx+1, 0:ny+1, ) | Impurity radiation due to individual impurity charge state [W/m3] |
| pwrze(0:nx+1, 0:ny+1) | Electron energy loss due to impurities [W/m3] |
| pradimp(0:nisp) | Domain-integrated radiation loss for each impurity charge state |
|  |  |
|  |  |
| vol(0:nx+1, 0:ny+1)\*eqp(0:nx+1, 0:ny+1)\* (te(0:nx+1, 0:ny+1)-ti(0:nx+1, 0:ny+1)) | Equipartition heat between ions and electrons |
| wjdote(0:nx+1, 0:ny+1) | **E**x**J** Joule heating of electrons, in [W] |
|  |  |
|  |  |

**Gradients**

|  |  |
| --- | --- |
| gpix(0:nx+1,0:ny+1,1:nisp) [Pa/m] | Poloidal ion pressure gradient: [Pa/m] |
| gpiy(0:nx+1,0:ny+1,1:nisp) | Radial ion pressure gradient: [Pa/m] |
| gpex(0:nx+1,0:ny+1) | Poloidal electron pressure gradient: [Pa/m] |
| gpey(0:nx+1,0:ny+1) | Radial electron pressure gradient: [Pa/m] |
| gprx(0:nx+1,0:ny+1) | Poloidal total pressure gradient: [Pa/m] |
| gpry(0:nx+1,0:ny+1) | Radial total pressure gradient: [Pa/m] |
| gtex(0:nx+1,0:ny+1) | Poloidal electron temperature gradient: [J/m] |
| gtey(0:nx+1,0:ny+1) | Radial electron temperature gradient: [J/m] |
| gtix(0:nx+1,0:ny+1) | Poloidal ion temperature gradient: [J/m] |
| gtiy(0:nx+1,0:ny+1) | Radial ion temperature gradient: [J/m] |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Currents**

|  |  |
| --- | --- |
| fqx(0:nx+1, 0:ny+1) | Net poloidal current, east face: total ion -electrons |
| fnix(0:nx+1, 0:ny+1, 1:nisp) | Ion poloidal current, cell east face, in [part/s]:  pol = niuix |
| fngx(0:nx+1, 0:ny+1, 1:ngsp) | Neutral poloidal current, cell east face |
| fqy(0:nx+1, 0:ny+1) | Net radial current, north face: total ion – electrons |
| fniy(0:nx+1, 0:ny+1, 1:nisp) | Ion radial current, cell north face, in [part/s]:  rad = niuiy |
| fngy(0:nx+1, 0:ny+1, 1:ngsp) | Neutral radial current, cell north face |
| fngyso(0:nx+1,ngsp) | Gas input flux from igaso on outer wall |
| fngysi(0:nx+1,ngsp) | Gas input flux from igaso on inner wall |
| feex(0:nx+1, 0:ny+1) | Poloidal electron heat q(pol, e), in [J/s]:  Q pol,e = 5/2\*neuexTe – kex\*∂Te/∂x – 0.71 neTe Bx/B\*Jpar/ene |
| feey(0:nx+1, 0:ny+1) | Radial electron heat q(rad, e), in [J/s]:  Q rad = 5/2\*neueyTe – key\*∂Te/∂y |
|  |  |
| feix(0:nx+1, 0:ny+1) | Total poloidal ion heat q(pol, i), in [J/s]:  Q pol,i = 5/2\*niuixTi – kix\*∂Ti/∂x |
| feiy(0:nx+1, 0:ny+1) | Total radial ion heat q(rad, i), in [J/s]:  Q rad,i = 5/2\*niuiyTi – kiy\*∂Ti/∂y |
|  |  |
| hxce(0:nx+1, 0:ny+1) | Poloidal electron thermal conductivity: [1/m/s] |
| hyce(0:nx+1, 0:ny+1) | Radial electron thermal conductivity |
| hcxi(0:nx+1,0:ny+1) | Poloidal ion and neutral thermal conductivity, summed |
| hcyi(0:nx+1,0:ny+1) | Radial ion and neutral termal conductivity, summed |
| hcxn(0:nx+1,0:ny+1) | Poloidal neutral thermal conductivity |
| hcyn(0:nx+1,0:ny+1) | Radial neutral thermal conductivity |
|  |  |
|  |  |
|  |  |
|  |  |

**Fields**

|  |  |
| --- | --- |
| bpol(0:nx+1, 0:ny+1, 0:4) | Poloidal magnetic field at cell center: [T] |
| bphi(0:nx+1, 0:ny+1, 0:4) | Toroidal magnetic field |
| brad(0:nx+1, 0:ny+1, 0:4) | Radial magnetic field |
| b(0:nx+1, 0:ny+1, 0:4) | Total magnetic field |
| rbfbt(0:nx+1, 0:ny+1) | Ratio btor/btot |
| rrv(0:nx+1, 0:ny+1), rrv(0:nx+1, 0:ny+1) | Ratio bpol/btot (pitch angle), referenced at cell center / east(north) face |
|  |  |
| ex(0:nx+1, 0:ny+1) | Poloidal electric field: [V/m] |
| ey(0:nx+1, 0:ny+1) | Radial electric field: [V/m] |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Geometry variables**

|  |  |
| --- | --- |
| nx+1 | Number of poloidal cells including guard cells |
| ny+1 | Number of radial cells including guard cells |
| sx(0:nx+1, 0:ny+1) | Cell surface area in poloidal direction, east face |
| sy(0:nx+1, 0:ny+1) | Cell surface area in radial direction, north face |
| vol(0:nx+1, 0:ny+1) | Cell volume |
| ixmp | Poloidal index of outer midplane |
| ixtop | Poloidal index of surface opposite x-point |
| ixpt1 | Poloidal index of last private flux cell before cut on left |
| ixpt2 | Poloidal index of last core flux cell before cut on right |
| 0.5\*(ixtop-ixpt1+1) | Poloidal index of inner midplane |
| xcs | Poloidal coordinate of the center of the (ix,iy) primary cell just outside separatrix, from inner plate |
| xcwo | Poloidal coordinate of cell center on outer wall (ny+1) |
| xcwi | Poloidal coordinate of cell center on inner wall (iy=0) |
| xcpf | As xcwi, but distances along core surface set to zero |
| xfs | Coordinate of the x-face of the (ix,iy) primary cells just outside the separatrix |
| xfwo | Coordinate of the x-face along outer wall |
| xfwi/xfpf | Coordinate of the x-face along inner wall |
| yyc | Y-coordinate of the center of the (ix,iy) primary cell at the outer midplane |
| yyf | Coordinate of the y-face of the primary cells at the midplane |
| ~porter/uedge/post/calc\_lengths:  lpol | Poloidal length to cell center from inner plate |
| lparal | Parallel length to cell center from inner plate |
| lpolf | Poloidal length to cell face from inner plate |
| lparalf | Parallel length to cell face from inner plate |
| iysptrx | Radial index of cell just inside the separatrix, cell just outside: iysptrx+1 |
| rm(0:nx+1, 0:ny+1, 0:4) | Radius of cell center and vertices: 0=center, 1=lower left, 2=lower right, 3=upper left, 4=upper right |
| zm(0:nx+1, 0:ny+1, 0:4) | Elevation of cell center and vertices |
| zshift | DIII-D: zshift=-1.6 |
| xlim() | Radii of vessel structure |
| ylim() | Elevation of vessel structure |
| gxf | 1/(x-distance) between density centers |
| gyf | 1/(y-distance) between density centers |
| yylb | distance along inner plate |
| yyrb | distance along outer plate |
| ixm1(0:nx+1, 0:ny+1) | ixm1(nx,yx)=nx-1, for all yx. Essentially a way of expressing nx-1 in basis |
| gx(0:nx+1, 0:ny+1) | gx(x,y)=1/(pol\_coord(x,y)-pol\_coord(x-1,y)), essentially an expression for 1/dx |
| gy(0:nx+1, 0:ny+1) | similarly as above in radial direction |
| ixp1 | poloidal index ix+1 |
| ixm1 | poloidal index ix-1 |

**Components of the momentum equation (deuterium only?)**

First run ~porter/uedge/post/rdmomeq. Post-processor will calculate all existing components in the ion momentum equation. Appears to be calculated (averaged) for the cell borders.

|  |  |
| --- | --- |
| vxmom | Average momentum in the poloidal direction: mass\*density\*velocity^2 |
| vymom | Average momentum in the radial direction |
| xpmcon | Poloidal gradient in momentum flow |
| ypmcon | Radial gradient in momentum flow |
| etxvdx | Averaged viscocity in the poloidal direction: eta(ix)\*dv/dx |
| etyvdy | Averaged viscocity in the radial direction:  eta(iy)\*dv/dy |
| xpmvis | Divergence of velocity in the poloidal direction: d/dx of eta(ix)\*dv/dx |
| ypmvis | Divergence of velocity in the radial direction:  d/dy of eta(iy)\*dv/dy |
| prdx | Gradient of total pressure in poloidal direction:  Bpol/B \* d(pi+pe)/dx |
| cxmneut | Momentum loss due to ion-neutral friction (CX): m\*nn\*ni\*<sigma\*(vn-vi)> |
| ionmion | Momentum gain due to ionization:  m\*nn\*ne\*<sigma\*vn> |
| recmion | Momentum loss due to recombination:  m\*nn\*ni\*<sigma\*vi> |
| mom\_net | Net momentum:  prdx+xpmcon+xpmvis+ypmcon+ypmvis-cxmneut-ionmion-recmion |
| prd2 | Btor/B\* d(pi+pe)/dx |
| jr\_tot | -prd2/(b0(scale factor for magnetic field, =1)\*btot) |
|  |  |
|  |  |
|  |  |

**Input parameters - transport**

|  |  |
| --- | --- |
| difni() | Ion diffusion coefficient: [m2/s] |
| kye() | Radial electron heat diffusivity |
| kyi() | Radial ion heat diffusivity |
| travis() | Value of perpendicular viscocity |
| isbohmcalc | 1 = Calculate Bohm diff if facb... > 0  2 = Harmonic ave of Bohm, difni, etc  3 = D=difniv\*(B0/B)\*\*inbtdif, etc  bscalf =  ((0.5\*(btot(ixmp,iysptrx)/btot(ix,iy)) + 0.5 \* btot(ixmp,iysptrx)/btot(ix1,iy)))\*\*inbtdif) \* ((bpol(ixmp,iysptrx,3)+bpol(ixmp,iysptrx,4))/ (bpol(ix,iy,3)+bpol(ix,iy,4)+bpolmin))\*\* inbpdif  dif\_use(ix,iy) = difniv(iy)\*bscalf  difp\_use(ix,iy) = difprv(iy)\*bscalf  dif2\_use(ix,iy) = difniv2(iy)\*bscalf  tra\_use(ix,iy) = travisv(iy)\*bscalf  kye\_use(ix,iy) = kyev(iy)\*bscalf  kyi\_use(ix,iy) = kyiv(iy)\*bscalf  dutm\_use(ix,iy) = difutmv(iy)\*bscalf  vy\_use(ix,iy,1) = vconyv(iy)\*bscalf |
| inbtdif | Variation of D\_Bohm in toroidal field direction |
| inbpdif | Variation of D\_Bohm in poloidal field direction |
| kyev | Electron heat diffusivity with D\_Bohm scaling |
| kyiv | Ion heat diffusivity with D\_Bohm scaling |
| difniv | Ion diffusion coefficient with D\_Bohm scaling:  Turn on radially varying D (or kyi, kye):  isbohmcalc=3;inbtdif=0;inbpdif=0;fcdif=0.0 |
| travisv | Perpendicular viscocity with D\_Bohm scaling |
| istabon | Look-up table for hydrogenic rate coefficients |
| isimpon | switch for impurity model:  0 for no impurities  2 for fixed-fraction model  3 for average-impurity-ion model  4 for INEL multi-charge-state model  5 for Hirshman's reduced-ion model  6 for force-balance model |
|  |  |

**Input parameters – pumping and gas puffing**

|  |  |
| --- | --- |
| recycp | Ion recycling coefficient at plates (1=full recycling as CX neutrals) |
| recycw | recycling coefficient at side walls |
| albdso/albdsi | albedo at outer/inner gas source location (1=no pumping) |
| matwso/matwsi | material wall at outer/inner gas source location (<10): 1=on |
| nwsor | Number of source regions on each outer **and** inner wall (must be < 10) |
| issorlb | Measure source positions from left plate (1) or right plate (0); def. 1 |
| igspsoro/igspsori | Index of gas species for outer/inner wall sources (<10), e.g., 1=deuterium, 6=carbon |
| xgaso/xgasi | Location of outer/inner wall source |
| issoro/issori, iesoro/iesori | Starting/ending cell index for outer/inner sources |
| wgaso/wgasi | Total cosine widths of outer/inner wall gas sources, typically 1/gx width |
| igaso/igasi | Gas current from outer/inner wall (iy=ny+1, in A) |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Flux limits and force modifiers**

|  |  |
| --- | --- |
| alfkxi | Flux limit on kxi if |ti(ix+1)-ti(ix)|<alfkxi\*ti(ix) |
| alfkxe | Flux limit on kxe if |te(ix+1)-te(ix)|<alfkxe\*te(ix) |
| b0 | Scale factor for magnetic field |
| cfybf | Coefficient for Grad B drift in y-direction |
| cf2bf | Coefficient for Grad B drift in 2-direction |
|  |  |
| cfgti | Scale factor for ion thermal force |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Starting a new case**

The first step is to generate a grid (“a” and “g” EFIT files).

**Re-starting UEDGE from savefile (probname.svpfb)**

It usually assumed that the saved solution is indeed a solution. The RHS of the equations may therefore be evaluated by:

read restart file (will restore save file)

issfon=0;ftol=1.e20;exmain

It will come back with a large fnrm. Ignore that. Setting the parameter issfon=0 tells the code to not bother evaluating the Jacobian. Setting ftol to a large number tells the code to consider the problem to have converged no matter what the RHS evaluates to. Since you haven't even bothered to evaluate the Jacobian, it is not like to come back with a small residual for the RHS. The solution was converged when you saved it, you are simply transferring the saved variables into the active variables, and using those to evaluate fluxes, etc.

**Changing an input parameter, run case**

Routines: read\_rdinitdt, rdrundt

1. Start from a converged plasma state by using restart\_probname, and do exmain.

2. Change whatever variable you want, lyni, lyte, lyti, t\_wall, etc. First type "read rdinitdt" to define the variables needed for running time dependently. This will set dtreal to 1.e-9, which is probably smaller than needed. I suggest you re-set dtreal to 1.e-6 or, if you expect small changes, even 1.e-4.

3. After you have dtreal set, simply type "read rdrundt". This will try converging with the value of dtreal you set. If it succeeds, it will increase dtreal and try to take another time step. If it doesn't converge in 15 iterations, it will pause and ask you what to do. If it fails, you will probably want to decrease dtreal and try again.

I suggest you try starting with a fairly large dtreal since these problems run slowly, so it is best to start with as large a dtreal as possible in order to limit the number of time steps you have to take. If you are successful the code will run for a total of 1 second, then stop. If you reach this point, you can usually go to a steady state solution by typing:

dtreal=1.e20;icntnunk=0;ftol=1.e-8;itermx=30;exmain

**Save variables in portable file (savefile)**

Routine: read saveit.pfb

**Restart UEDGE case after it run out of iterations**

read reset; itermx=30 (or higher); exmain

**Case fails to converge when running time-dependently**

If fnrm sufficiently low (<1x10-4) then try the following trick:

* Set mfnksol=-3 (default is 3): change method of solving Newton equations, allow negative steps
* Set icntnunk=0 (default is 1): icntnunk
  + icntnunk: nksol continuation call flag
    - =1 tells nksol not to call the preconditioner routine, pset on the current call. In this case, nksol assumes that the preconditioner was evaulated on an earlier call, and is to be used for as many steps as it is successful on this call.
    - =0 tells nksol that this is not a continuation call. The preconditioner routine pset is called to evaluate and factor the Jacobian matrix.
* Set del=1e-9: (default ): fractional change for finite diffs
* Set itermx=30: number of allowed iterations
* exmain
* If still fails to converge (solution sometimes oscillates):
  + Set mfnksol=3
  + Set adjf1=1.0 (default 1.2): if mfnksol=3 glob strat, frnm\_new/adjf1>=fnrm\_old

read idtroub: gives cell and variable with larges yldot\*sfscal

Plot time history of this parameter: plot var\_stor(1:i\_stor-1,,,) tim\_stor(1:i\_stor-1)

Set dtreal to 1e-5 or even 1e-6, exmain

From UEDGE Vers\_4.40\_beta\_mds: Freeze hydrogen state variables, then solve for carbon only, dtreal=1e-5:

* isnion(1:2)=0; isupon=0; isteon=0; istion=0; isphion=0; isphiofft=1

Or freeze carbon and solve for hydrogen

* isnion(3:8)=0; isngon(2)=0

**Restore and reset state variables (nis, ngs, tes, tis, phis) from time-dependent save file (pftstor\_probname)**

* read pftstor\_probname
* find last index in tim\_stor
* reset state variables: nis=ni\_stor(tim\_index,,,), ngs=ng\_stor(tim\_index,,,)

**Turn off potential and currents (i.e., turn off drifts)**

* Eliminate “Currents and potentials” block in input deck
* May need to set phis=40. to make sure that potentials are disregarded

**Estimate core plasma fueling due to beams**

* At standard beam voltage: 1 MW = 20A

**Useful BASIS post-processor routines**

|  |  |
| --- | --- |
| rdforce.imp\_groth | Calculates forces due to hydrogen drag, temperature gradients, viscosity on all impurity ions |
| write\_impforce | Writes parallel force balance components into tab-delimited file for a specific iy=iysptrx+**iydelta** |
| write\_par\_impforce | Like write\_impforce, but prompts user to input ionspecies and iy, also stores impurity pressure and impurity pressure gradient |
| write\_wall\_source | Writes total wall source of neutral hydrogen and carbon; output is four files: outer and inner wall, and outer and inner plate |
| balancepp | Calculates various particle and heat fluxes, write data to file named bal\_probname |
| **Tab-delimited file, profile data** | |
| expnoplot\_nog | Various output files, including div\_probname: plate profiles; requires balancepp to be run prior |
| write\_omp\_profiles | Outer midplane profiles: omp\_probname |
| write\_radflux\_polprof | Tab-delimited file of radial fluxes and currents across iy, poloidal profile: total, GradB, ExB, diffusive; prompts for ionspecies: RadFlux\_Sp#\_iix#\_probname |
| write\_polflux\_radprof | Tab-delimited file of poloidal fluxes and currents across ix, radial profile: total, GradB, ExB, diffusive; prompts for ionspecies; PolFlux\_Sp#\_iix#\_probname |
| write\_polflux\_polprof | Tab-delimited file of poloidal fluxes and currents along iy, poloidal profile: total, GradB, ExB, diffusive; prompts for ionspecies; PolFlux\_Sp#\_iiy#\_probname |
| write\_divflux | Power and particle fluxes to the divertor plate: div\_probname, dpwr\_probname |
| write\_pol\_mombal | Poloidal profile of compents in momentum balance equation, and deuterium pressure, pressure gradient, prompts for iy |
|  |  |
| **Vector plot** | |
| rdpltvec | Prepare to plot ion flux vectors |
| rdfluxt | Calculate poloidal ion flux |
| plot\_ionvec\_species | Plots poloidal flux diagram (vector) for individual ion species |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Useful UNIX commands**

|  |  |
| --- | --- |
| find . -exec grep pltvec {} \; -print | more | Find recursively, current directory, exec = excute grep, curly bracket = place holder for file name, \; end of command, print filename only if grep is true |
| tar –vxf filename | un-tar filename, -v=verbose, x=extract, f=file |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Useful BEOWULF commands**

brun –x `alias xu\*\*\*\*\*` (remember the inverted single quotes)

busers

bjobs

If the case hangs up and you want to kill the beowulf session, do the following

* bjobs: will show the attached and detach runs – determine on what mfelinux machine the case is/was running, determine brun id
* screen –x brun id – shows what is currently running
* to leave ‘screen’ press ctrl-a ctrl-d
* ssh to that mfelinux machine
* ps –ef | grep username – note the run id
* kill -9 run id – kills the job

To kill a dead vnc session, do the following

* vncserver –kill :1
* rm ~/.vnc/fepilx:1.pid

**Useful post-processors**

write\_corrfactors: calculates various loss factors, most probably not correctly but is a nice display of how to use BASIS

write\_balancepp\_andreas: exports the most common uedge parameters to .dat files. Corrected in order to easily add/remove cases

write\_momeq: writes the various terms of the momentum equations with separated volumetric sources

write\_basicparams: writes the temperatures, densities and pressures at the separatrix upstream, x-point and target locations

**Setting up the window for UEDGE**

* UEDGE is programmed in BASIS, and does not understand some of the modern definitions of IP and ports.
* When logged into triton, run:
* printenv DISPLAY: displays hostip:port.0
* nslookup hostip: Dipslays various parameters. Important is "Non-authoritative answer: Adress: AAA.BBB.CCC.DDD"
* export DISPLAY=AAA.BBB.CCC.DDD:port
* This sets the display so windows can be opened in UEDGE

**UEDGE in short - Simplified geometry**

UEDGE is quite poorly documented, most of the critical variables being found in this document. If you need information about a variable that is not here, there are a few choices: do a list of the variable. This gives you an idea of what it is. Print the variable to see the values. Look for the variable in the bbb.v file. If all above fails: guess what it might do. The bbb.v file contains most, if not all relevant variables. Those not accounted for there or in the initialization files are most likely scaling factors.

**Geometry**

The simplified geometry consists of a rectangle with 64 cells in the x-direction and 16 in the y-direction. There are also guard cells surrounding the whole rectangle, making all geometric variables 66\*18 in size. The guard cells are situated at the poloidal and radial indices of 0 and 65, and 0 and 17 respectively. These should be left out in all data processing. The poloidal and radial positions of each cell is given by the corresponding index of the variables xcs and yyc respectively.

**Commands**

**read restartfile**

* Reads the restart file
* Sets up the case according to the parameters
* Loads the save-file specified in the initialization file

**exmain**

* Runs the case.
* Uses the save-file as start for the case, if no alterations has been made
* If alterations has been made since the read-command, UEDGE will use the last successfully converged case as starting point
* Progress needs to be saved when changing cases, or UEDGE will resume from the save-file

**win on windowname**

* Opens new window with name windowname
* Change to this window by using the same command
* clear active window with the command nf
* close window by using command: win close windowname

**plot variable(x,y) xaxis color=rainbow**

* Plots the variable against the x-axis.
* Uses different color for each plot command
* Plots into active window
* x poloidal, y radial cells
* To plot the poloidal or radial direction, replace x and y respectively with a blank
* To coose range from a to b replace x or y with a:b
* To choose all in poloidal or radial direction, replace x or y respectively with :
* If you choose to plot a poloidal profile, use xaxis=xcs
* If you choose to plot a poloidal profile, use xaxis=yyc
* For example plot ne(1:64,:) xcs(1:64) color=rainbow plots all the poloidal density profiles starting from the inner radial guard cell, ending in the outer radial guard cell.
* Possible to plot two variables against each other

**frame xmin,xmax,ymin,ymax**

* Set the graph to display the interval xmin<x<xmax and ymin<y<ymax

**read rdinitdt**

* Sets up the variables for running a rdrundt\_slab
* Used when looking for a steady state solution for a new case
* Change the dtreal to largest possible value for convergence (find this value by trying different dtreal and the running an exmain. Use largest value when the exmain converge)
* If you have trouble getting convergence at dtreal=1e-9 it might be that you are outside the radius of convergence. Try reversing the changes made to the initialisation file, and bring the values toward the targets stepwise, running a new rdrundt\_slab between each change.

**read rdrundt\_slab**

* Program for iterating a steady state solution for a new case
* Increases Time from dtreal towards e0
* Increments of dTime
* Set number of iterations: If the case does not converge in this number of iterations, a "troublemaker equation" is run. This essentially lowers the dtreal.
* If a case is not going to converge, the troublemaker will occur multiple times, lowering dtreal.
* The rdrundt\_slab iterates at maximum a few thousand times. Thus if dtreal is more then 4 orders of magnitude smaller than dTime, it is safe to say that the case will not converge.
* To abort the current operation: Ctrl+c to open debug and "abort" to abort operation
* See above for instructions on how to use rdrundt\_slab-command

**read saveit.slab**

* Run the lines mentioned earlier in the text before this command
* Saves the current converged solution to a save-file
* Save file will be named according to the **variable specified in the** **setup file**
* Make sure that this variable has its own name, or it will overwrite the save-file in the variable!
* If you get an error running the saveit.slab script, it is due to the pure plasma! The original saveit.slab is set to write impurities, but will crash when there are none. The saveit.slab can be opened in emacs, and the script runs several other files. In these files, comment out the problematic rows. Alternatively, rename the files in your directory so they are saved for later use and copy the corresponding files from holma2/uedge/post/... These are corrected for pure plasmas

**list varname**

* Displays the most basic properties of varname: size, unit, where it is stored etc.
* Useful for figuring out what a variable is and how it looks

**end**

* quit UEDGE

**Post-processors**

As mentioned earlier, UEDGE runs on BASIS: an ancient language. The only guide to the language I have ever found is here: https://wci.llnl.gov/codes/basis/manual/ A local copy of it can be useful as it will probably go down sooner or later. All scripts and post-processors need to be written in basis, and follow the specifications. Most notably, BASIS executes the loop to completion before running any read-commands. This means **loops opening files do not work**. Here follows some tips and guides for my personal post-processors. All scripts and post-processors are used with the command "read filename".

**write\_balancepp\_10\_andreas**

* Compiles the most basic data into three .dat files
* Case-dependent data, does not take geometry etc into account.
* Is altered from the original write\_balancepp\_10 so that jx cases are read and saved
* jx is the first variable in the script. This defines how many cases are saved. Alter this and the string prob to read more or fewer cases
* prob is a queue of strings specifying the cases to be read. These names are appended to restart\_, so if you want to save the data from case restart\_A, the prob-entry should read "A".
* Also make sure that there are sufficient writing segments: the command lines between the dashed lines. jy specifies which case is currently working on, so jy.leq.jx. If there are too few writing segments, add segments and change jy.
* Reads the post-processor balancepp

**balancepp**

* Run as part of write\_balancepp\*
* If you get an error while tunning the write\_balancepp\* it is due to this file trying to write impurities. If you have a pure plasma this will crash the program. Copy the same file from holma2/uedge/post or make the same alterations to your own balancepp. Remeber to backup the old one.

**write\_momeq\_andreas**

* Writes the momentum loss factors. A more refined script than the write\_balancepp\_10\_andreas. I recommend to use this as a model when writing new scripts.
* case defines the number of cases
* ycoord what radial flux tube is considered
* prob a string with case names
* fname what the .dat file will be named
* varname what string will be appended to the variables
* Changing the names will allow for making multiple files quickly, which can be imported into matlab simultaneously without renaming any parameters.

**write\_pol\_profile.holm**

* Writes variable values in the poloidal direction for the radial flux tube ycoord from one file into a .dat file.
* Works as the above, but does writes raw data