`BBID_solve.c` documentation

Niclas Moldenhauer
`niclas.moldenhauer@uni-jena.de`

September 14, 2013

The purpose of `BBID_solve.c` is to compute constraint solved initial data for neutron star binaries on eccentric orbits. The physical background is not explained here.

Contents

1 Basic structure

Essentially, the code consists of seven different functions. The main function, which calls all other functions, is

- `int BBID_solve(tL* level)`

It calls a function `multigrid(...)`, which expects as argument two functions to specify the (non)linear Poisson operator to solve the constraint equations. These are given by the functions

- `void LBBID_GS(tL* level, tVarList *vlv, tVarList *vlu)`
- `void LBBID_nonlin(tL* level, tVarList *vllu, tVarList *vlu)`

The remaining functions

- `double massintegral(tL* level)`
- `void find_constants(tL* level)`
- `void compute_eta(tL* level)`
- `void compute_sources(tL* level)`

are used to calculate certain quantities during the iterations. The general flow of the program can be seen in the diagram.

2 In detail

2.1 `BBID_solve()` - the main function

Before the `BBID_solve()` is called, `bam` computes the superimposed TOV solution. In the beginning of our function, after the declaration, all variables are initialized with these values. Note that `omegaq` denotes $\psi^{-n} - 1$ and `apsiq` denotes $\alpha\psi - 1$. At some points we make use of `psiq`, which represents $\psi - 1$. The subtraction of one is necessary because of the way `bam` handles the boundaries.

Then

```
for (iter=0; iter <=itmax; iter++)
```

starts the main iteration loop. As a first step, the constants `Econst_0`, `Econst_1` and `Omega` are computed by `find_constants(...)`. This happens only on the finest grid. To compute `Omega`, we use the integrated Euler equation, derivated with respect to x. This is solved for Ω and evaluated at the given center $x_{0,1}$ of the star (we make use of the vanishing derivative of the enthalpy h'). Afterwards, we use the integrated Euler equation at the center and demand the central value of the enthalpy to be the given parameter `eta0`.

2.2 `compute_eta()`

According to the constants, we compute a new density profile by using again the integrated Euler equation. This is done within the function `compute_eta(...)` by evaluating (in the corotating case)

```
eta[ijk] =
soft*((xp[ijk]>=0 ? E_const0a:E_const0b)/
  sqrt(pow(alpha[ijk],2)-pow(1+psiq[ijk],4)*
    (pow(-Omega*yp[ijk]+betax[ijk],2)
    +pow(Omega*xp[ijk]+betay[ijk],2)
    +pow(betaz[ijk],2)))
  -1)
+(1-soft)*eta[ijk];
```

One can see that softening is applied. The function also searches for negative values of `eta` and sets them to zero, as well as all points which are further away from the initial star than the initial radius plus 1M. Afterwards, `ut` is computed.

2.3 the multigrid solver

In the next step, we make a temporary copy of the elliptic variables to apply softening after the solve and then we call the multigrid solver by

```
multigrid(level, vlu, vlf, vlv, vlc,
               itmax, tol, &normres,
               LBBID_nonlin, LBBID_GS);
```

where `vlc`, `vlu` and `vlv` contain the variable lists of our elliptic variables and the constants (`rhoH, S, Si, eta, ut`).

Therefore we have to specify the functions `void LBBID_GS(...)` and `void LBBID_nonlin(...)`, which look very similar, but with some small differences defining the operator. Both functions start by allocating the needed variables and defining the finite differencing dx terms. Then the sources `S`, `Sx`, `Sy`, `Sz` and `rhoH` are computed by using `compute_sources()`. Now in `LBBID_GS()`, the left-hand side of the constraint equations (in the form $\mathcal{H} = 0$) is defined as `lu` and the updated values of the quantity X is defined as $v_X = u_X + (f_X - lu)/lii$, where u_X is the old value and $lii = \partial lu/\partial X$.
The `LBBID_nonlin()` function deviates from this, since it just computes the `lu` for each quantity. The implemented equations in each case are

```
lu = cc * u_omegaq[ccc] +
     cx * (u_omegaq[mcc] + u_omegaq[pcc]) +
     cy * (u_omegaq[cmc] + u_omegaq[cpc]) +
     cz * (u_omegaq[ccm] + u_omegaq[ccp])
       - (n+1)/n*(pow(cx_1*(u_omegaq[pcc]-u_omegaq[mcc]),2)
                 +pow(cy_1*(u_omegaq[cpc]-u_omegaq[cmc]),2)
                 +pow(cz_1*(u_omegaq[ccp]-u_omegaq[ccm]),2))
         *pow(1+u_omegaq[ccc],-1)
       - 2*PI*n*rhoH[ccc]*pow(1+u_omegaq[ccc],-4./n+1)
       - n/8. * pow(1+u_omegaq[ccc],8./n+1)*AA[ccc];
```

which should physically correspond to the rescaled Hamiltonian constraint equation

$$\Delta\Omega - \frac{n+1}{n}\left(\nabla\Omega\right)^2 \cdot \Omega^{-1} - 2\pi n \rho_H \Omega^{-4/n+1} - \frac{n}{8}\Omega^{8/n+1}\bar{A}^{ij}\bar{A}_{ij} \tag{1}$$

And the second equation we solve, is the lapse equation

```
lu = cc * u_apsiq[ccc] +
       cx * (u_apsiq[mcc] + u_apsiq[pcc]) +
       cy * (u_apsiq[cmc] + u_apsiq[cpc]) +
       cz * (u_apsiq[ccm] + u_apsiq[ccp]) -
       (1+u_apsiq[ccc])*(0.875*pow(1+u_omegaq[ccc],8./n)*AA[ccc]
          +2*PI*pow(1+u_omegaq[ccc],-4./n)*(rhoH[ccc]+2*S[ccc]));
```

which is the discretized version of

$$\Delta(\alpha\psi) - \alpha\psi\frac{7}{8}\left(\Omega^{8/n}\bar{A}^{ij}\bar{A}_{ij} + 2\pi\Omega^{-4/n}(\rho_H + 2S)\right) \tag{2}$$

The momentum constraint equations are discretized like (here e.g. the x component)

```
lu = cc * u_betax[ccc] +
     cx * (u_betax[mcc] + u_betax[pcc]) +
     cy * (u_betax[cmc] + u_betax[cpc]) +
     cz * (u_betax[ccm] + u_betax[ccp]) +
     1./3. * ( cx*(u_betax[pcc]-2.*u_betax[ccc]+u_betax[mcc])
       +  cx_1*cy_1*(u_betay[ppc]-u_betay[pmc]-u_betay[mpc]+u_betay[mmc])
       +  cx_1*cz_1*(u_betaz[pcp]-u_betaz[pcm]-u_betaz[mcp]+u_betaz[mcm]))
     -Sx[ccc];
```

which is the same for the other components, apart from a change in the source term S^i. This equation corresponds to

$$\Delta\beta^i + \frac{1}{3}\partial^i\partial_j\beta^j - S^i \tag{3}$$

One should note here that for convenience the whole source term is included in S^i.

2.4 compute_sources

This can be seen in the `compute_sources()` function, where we compute for example in the corotating case

```
Sx[ccc] = 2.*pow(1+psiq[ccc],-7)*
    ( Axx*(dapsix-7.*alpha*dpsix)
    + Axy*(dapsiy-7.*alpha*dpsiy)
    + Axz*(dapsiz-7*alpha*dpsiz))
  + 16.*PI*(epsilon+p)*alpha*alpha*ut[ccc]*ut[ccc]*
     pow(1+psiq[ccc],4)*(betax[ccc]-Omega*yp[ijk]);
```

which comes from

$$S^x = 2\psi^{-7}\bar{A}^{xj}(\partial_j(\alpha\psi) - 7\alpha\partial_j\psi) + 16\pi(\epsilon+p)(\alpha u^t)^2\psi^4(\beta^x - \Omega y) \tag{4}$$

(*Note*: Before we compute the source we calculate ψ from Ω. The Ω in this source term is not the elliptic variable, but the orbital frequency, which is distinguished by the capital O.)
For that computation, the traceless part of the extrinsic curvature is needed, which is

$$A^{ij} = \frac{\psi^6}{2\alpha}(\partial^i\beta^j + \partial^j\beta^j - \frac{2}{3}\bar{\gamma}^{ij}\partial_k\beta^k) \tag{5}$$

In the C code this yields for example

```
Azz = 0.5*pow(1+psiq[ccc],6)/alpha*(4/3*dbetazz-2/3*(dbetayy+dbetaxx));
Axy = 0.5*pow(1+psiq[ccc],6)/alpha*(dbetayx+dbetaxy);
```

And the trace can be computed as

```
AA[ccc]  = (Axx*Axx+Ayy*Ayy+Azz*Azz+2.*(Axy*Axy+Axz*Axz+Ayz*Ayz));
```

The other source terms are computed straight-forward from the formulas

$$S = (\epsilon+p)((\alpha u^t)^2 - 1) + 3p \tag{6}$$

$$\rho_H = \epsilon + (\epsilon+p)((\alpha u^t)^2 - 1) \tag{7}$$

```
S[ccc]     = ((epsilon+p)*(alpha*alpha*ut[ccc]*ut[ccc]-1)+ 3.*p);
rhoH[ccc]  = (epsilon + (epsilon+p)*(pow(alpha*ut[ccc],2)-1));
```

Where we use the following relations from the polytropic equations of state

$$\rho_0 = \left(\frac{\eta_0}{\kappa(1+n)}\right)^n \tag{8}$$

$$p = \rho_0\left(\frac{\eta}{\eta_0}\right)^{n+1}\frac{\eta_0}{1+n} \tag{9}$$

$$\epsilon = \left(\frac{\eta}{\eta_0}\right)^n(1 + \frac{n}{1+n}\eta) \tag{10}$$

```
rho0     = pow(eta0/(K*(1.+poly)),poly);
p        = rho0*pow(eta[ijk]/eta0,poly+1)*eta0/(1+poly);
epsilon = rho0*pow(eta[ijk]/eta0,poly)*(1+poly/(1+poly)*eta[ijk]);
```

```
BBID_solve( )
    initialize values as superimposed TOV
    Iterate until convergence
        compute eta
        compute constants $\varepsilon$ and $\Omega$
        Using multigrid solver
            linear Gauss-Seidel (LBBID_GS)
                compute_sources
                compute linear operator
            nonlinear step (LBBID_nonlin)
                compute_sources
                compute nonlinear operator
```