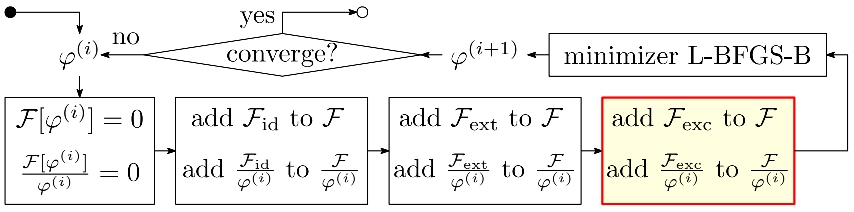
1 Algorithms and Branches

As discussed in section **??**, in code MDFT, we evaluate the functional $\mathcal{F}[\varphi]$ and its gradient $\dfrac{\delta\mathcal{F}[\varphi]}{\delta\varphi}$ in each iteration.



yes
no
$\varphi^{(i)}$
converge?
$\varphi^{(i+1)}$
minimizer L-BFGS-B
$\mathcal{F}[\varphi^{(i)}] = 0$
$\frac{\mathcal{F}[\varphi^{(i)}]}{\varphi^{(i)}} = 0$
add $\mathcal{F}_{\text{id}}$ to $\mathcal{F}$
add $\frac{\mathcal{F}_{\text{id}}}{\varphi^{(i)}}$ to $\frac{\mathcal{F}}{\varphi^{(i)}}$
add $\mathcal{F}_{\text{ext}}$ to $\mathcal{F}$
add $\frac{\mathcal{F}_{\text{ext}}}{\varphi^{(i)}}$ to $\frac{\mathcal{F}}{\varphi^{(i)}}$
add $\mathcal{F}_{\text{exc}}$ to $\mathcal{F}$
add $\frac{\mathcal{F}_{\text{exc}}}{\varphi^{(i)}}$ to $\frac{\mathcal{F}}{\varphi^{(i)}}$

Figure 1.1: Process "find equilibrium density" in MDFT. After the "initiation" process, the flow chart begins at the black point. The three terms of functional and their gradients are accumulated in order. The process end at the white point, which links to "output" process.

In this chapter, we present all the algorithms to evaluate the excess functional $\mathcal{F}_{\text{exc}}[\rho(\mathbf{r},\mathbf{\Omega})]$, knowing that

$$\rho(\mathbf{r},\mathbf{\Omega}) = \rho_0\varphi^2(\mathbf{r},\mathbf{\Omega}) \tag{1.1}$$

According to the commutativity of operations (see §**??**), the only possible algorithms to evaluate $\gamma(\mathbf{r},\mathbf{\Omega})$ from $\Delta\rho(\mathbf{r},\mathbf{\Omega})$ are shown in the figure 1.2.

$$\begin{tikzcd}
\boxed{\Delta\rho(\mathbf{r},\mathbf{\Omega})} \arrow[r, "b"] \arrow[d, "a"] & \boxed{\Delta\hat\rho(\mathbf{k},\mathbf{\Omega})} \arrow[r, "e"] \arrow[d, "d"] \arrow[rrr, bend left, "h"] & \boxed{\Delta\hat\rho(\mathbf{k},\omega)} \arrow[r, "i"] \arrow[d, "g"] & \boxed{\hat\gamma(\mathbf{k},\omega)} \arrow[r, "m"] & \boxed{\hat\gamma(\mathbf{k},\mathbf{\Omega})} \arrow[r, "p"] & \boxed{\gamma(\mathbf{r},\mathbf{\Omega})} \\
\boxed{\Delta\rho^m_{\mu'\mu}(\mathbf{r})} \arrow[r, "c"] & \boxed{\Delta\hat\rho^m_{\mu'\mu}(\mathbf{k})} \arrow[r, "f"] \arrow[rrr, bend right, "k"] & \boxed{\Delta\hat\rho'^m_{\chi\mu}(\mathbf{k})} \arrow[r, "j"] & \boxed{\hat\gamma'^m_{\chi\mu}(\mathbf{k})} \arrow[r, "n"] \arrow[u, "l"] & \boxed{\hat\gamma^m_{\mu'\mu}(\mathbf{k})} \arrow[r, "q"] \arrow[u, "o"] & \boxed{\gamma^m_{\mu'\mu}(\mathbf{r})} \arrow[u, "r"]
\end{tikzcd}$$

Figure 1.2: Possible algorithms for γ evaluation

Several branches are built to test and compare between algorithms, which are shown below in table 1.1 and will be detailed in the following context.

These branches should give numerically the same result in certain conditions that will be discussed in later sections.

1.1 Branches "naive"

Branches **naive** are the algorithms mentioned in section **??**, which go through the path

$$(b) \to (h) \to (p)$$

in figure 1.2, calculating directly $\hat\gamma(\mathbf{k},\mathbf{\Omega})$ from $\Delta\hat\rho(\mathbf{k},\mathbf{\Omega})$. The difference between branches is the way to calculate $\hat c(\mathbf{k},\mathbf{\Omega_1},\mathbf{\Omega_2})$. Branch **naive_standard** use $c^{mn}_{\mu\nu,\chi}(k)$ as input **DCF!**. Branch **naive_zero-order** and **naive_interpolation** use $\hat c(k,\boldsymbol{\omega_1},\boldsymbol{\omega_2})$ with zero-order and linear interpolation, where the former is rejected in the implementation due to a lack of precision (appendix **??**).

METHOD	SUB-METHOD	DESCRIPTION	THEORY
reference	dipole	calculate $n(r)$ and $P(r)$ separately	§?? [ref]
naive	standard	use $c^{mn}_{\mu\nu,\chi}(k)$ as input **DCF!**	§??
	zero-order	use $\hat{c}(k, \boldsymbol{\omega_1}, \boldsymbol{\omega_2})$ and take the nearest point	§??
	interpolation	use $\hat{c}(k, \boldsymbol{\omega_1}, \boldsymbol{\omega_2})$ with linear interpolation	§??
	dipole	use c_S, c_Δ, c_D issue from [ref]	§??
	nmax1	use c_S, c_Δ, c_D, $c_\pm$ issue from [**puibasset_bridge_2012**]	§??
convolution	standard	algorithm with symmetry reduction	§??
	asymm	algorithm without symmetry reduction	§??
	pure_angular	inverse **FFT!** and **FGSHT!**	§1

Table 1.1: Branch option in MDFT

1.2 BRANCHES "CONVOLUTION"

Branches **`convolution_asymm`** and **`convolution_standard`** are operational algorithms of angular convolution show in section **??**, which go through the path

$$(a) \to (c) \to (f) \to (j) \to (n) \to (q) \to (r)$$

Branches **`convolution_asymm`** uses the original operational algorithm (§**??**) without symmetry reduction (§**??**), and **`convolution_standard`** with it.

Branch **`convolution_pure_angular`** goes through the path

$$(b) \to (d) \to (f) \to (j) \to (n) \to (o) \to (p)$$

which inverts the first and last two steps of the two algorithms mentioned above.

1.3 TESTING BRANCHES FOR $n_{\max}$=1

Branches **`naive_dipole`**, **`naive_nmax1`** pass by $(b) \to (h) \to (p)$, using **DCF!** separately of the references [ref] and [**puibasset_bridge_2012**], whose slight difference is shown in §**??**. Branch **`reference_dipole`** use **DCF!** in [ref], which is the original method in **MDFT!** to calculate $\mathcal{F}_{\text{exc}}$ via multipole expansion. In addition with branch **`convolution_standard`**, which can also use the two **DCF!** mentioned above, a test of validation can be performed, which should in any case be exactly the same numerically if the same **DCF!** is used.

1.4 OTHER PATHS

Considering the necessity, other paths such as those passing by (i) and (k) are only built for local test usage (c. f. discussion in following sections).