Potfit Documentation

Cheat Sheet 1 - Force Code

Introduction

The goal of these cheat sheets is to give minimal sample developer code to help others extend Potfit. In this section, we focus on the force codes, such as *force_pair.c* and *force_meam.c*.

```
#include "potfit.h"

#include "chempot.h"
#if defined(MPI)
#include "mpi_utils.h"
#endif
#include "force.h"
#include "functions.h"
#include "potential_input.h"
#include "splines.h"
#include "utils.h"


// Inf loop
while(1){


  /* If we have an analytical potential with NO MPI, then we
     can directly call these functions*/
#if defined(APOT) && !defined(MPI)
  // If it's an analytic function
  if (g_pot.format_type == POTENTIAL_FORMAT_ANALYTIC) {
    // Check if the given parameters are valid.  Ex, is R > S?
    apot_check_params(xi_opt);

    // Now, update g_pot.calc_pot.table from g_pot.opt_pot.table, including globals
    update_calc_table(xi_opt, xi, 0);
  }
#endif  // APOT && !MPI

  /*Now, if instead we have MPI but NOT ANALYTICAL POTENTIAL, we just
    need to broadcast g_pot.calc_pot across the processors.*/
#if defined(MPI)
#if !defined(APOT)
  // exchange potential and flag value
  MPI_Bcast(xi, g_pot.calc_pot.len, MPI_DOUBLE, 0, MPI_COMM_WORLD);
#endif  // !APOT

  // Broadcast the flag across processors
  MPI_Bcast(&flag, 1, MPI_INT, 0, MPI_COMM_WORLD);
  if (flag == 1)
    break; // Exception: flag 1 means clean up

  /* HOWEVER!  If we DO have an analytical potential, then we need to verify the parameters
     are correct (only once, so we do if to mpi id == 0), but we need to broadcast the values
```

```
     to all the MPI_Bcast to update.  That is, we first check if xi_opt are okay (if not, fix)
     in the apot_check_params function, then we update the values across the other processors.*/
#if defined(APOT)
  if (g_mpi.myid == 0)
    apot_check_params(xi_opt);
  MPI_Bcast(xi_opt, g_calc.ndimtot, MPI_DOUBLE, 0, MPI_COMM_WORLD);
  update_calc_table(xi_opt, xi, 0);
#else   // APOT

  // This is what takes care of the broadcasting.
  // if flag == 2 then the potential parameters have changed -> sync
  if (flag == 2)
    potsync();
#endif  // APOT
#endif  // MPI




  // Loop over configurations
  for (int config_idx = g_mpi.firstconf; config_idx < g_mpi.firstconf + g_mpi.myconf; config_idx++) {

    // Loop over atoms in this configuration
    for (int atom_idx = 0; atom_idx < g_config.inconf[config_idx]; atom_idx++) {

      // At this point, forces can be found via the following
      double fx = forces[3*(g_config.cnfstart[config_idx] + atom_idx) + 0];
      double fy = forces[3*(g_config.cnfstart[config_idx] + atom_idx) + 1];
      double fz = forces[3*(g_config.cnfstart[config_idx] + atom_idx) + 2];

    }

    // Now, say we wanted to loop over atoms, and neighbors, with pointers to the atom/neighbor:
    for (int atom_idx = 0; atom_idx < g_config.inconf[config_idx]; atom_idx++) {
      // Grab an atom pointer
      atom_t* atom = g_config.conf_atoms + atom_idx + g_config.cnfstart[config_idx] - g_mpi.firstatom;

      // Loop over all neighbors
      for (int neigh_idx = 0; neigh_idx < atom->num_neigh; neigh_idx++) {
        // Grab a neighbor pointer
        neigh_t* neigh = atom->neigh + neigh_idx;

        // Now, we get some properties.  For instance, how far neigh is to atom:
        double dist = neigh->r;
      }

    }

  }
```

```
    // A convenient way to combine error_sum and forces across processors if MPI was used
    gather_forces(&error_sum, forces);

    // root process exits this function now
    if (g_mpi.myid == 0) {
      // Increase function call counter
      g_calc.fcalls++;
      if (isnan(error_sum)) {
#if defined(DEBUG)
        printf("\n--> Force is nan! <--\n\n");
#endif  // DEBUG
        return 10e10;
      } else
        return error_sum;
    }


}
```