gfp_gaussian_process

Compile

```
cd src; make
```

Run

```
cd bin
./gfp_gaussian <csv file created by moma>
```

The columns that are taken from the csv file can be set by modifying `csv_config.txt`, although defaults are set.
The parameter space is defined in `parameter_bound.txt`.

Notes gfp_gaussian_process

Libraries

- Minimization: nlopt
  - can be installed via cmake
  - can be statically compiled easily
- Linear algebra: Eigen
  - available via modules

TODO: Likelihood calculation

- [x] mean and covariance matrix at cell division, log_likelihood, posterior
- [x] set initial guess for mean and covariance matrix, (not really tested yet)
- [ ] prior distribution for the next cell state: calculation of mean and covariance matrix

```
void mean_cov_model(MOMAdata &cell,
                    double t, double ml,
                    double gl, double sl2,
                    double mq, double gq,
                    double sq2, double b);
```

- [ ] get a simulated data set as reference and test everything
- [ ] decide on output

Structure

```
CSVconfig config("csv_config.txt");
Parameter_set params("parameter_bounds.txt");

std::cout << params << "\n";

// Read data
std::string infile = argv[1];
if(! std::__fs::filesystem::exists(infile)){
    std::cout << "File " << infile << " not found! \nQuit" << std::endl;
    return 0;
}
std::vector<MOMAdata> cells =  getData(infile,
                                       config.time_col,
                                       config.length_col,
                                       config.fp_col,
                                       config.delm);


// genealogy
build_cell_genealogy(cells);
print_cells(cells);

// minimization for tree starting from cells[0]
minimize_wrapper(&total_likelihood, cells[0], params);

// DEMO on how this works for all trees
// get the "trees" starting from all root cells
std::vector<MOMAdata *> root_cells = get_roots(cells);
std::vector<double> dummy_params;
for(long j=0; j<root_cells.size(); ++j){
    print_generation_tree(dummy_params, *root_cells[j]);
}
```

Likelihhod Calculation

- apply function recursively
- every cell is accessed once and after its parent is calculated

```
    /* applies the function func to the cell cell and the other cells in the genealogy
    * such that the parent cell has already been accessed when the function is applied
    * to the cell.
    *
    * Example (number implies the order in which)
    * _____________________________________________________

              1
            /   \
           2     5
         /   \     \
        3     4     6

    * _____________________________________________________
    */
void likelihood_recr(const std::vector<double> &        params_vec, MOMAdata *cell, doul
    /*
    * Recursive implementation that applies the function func to every cell in the genea
    * not meant to be called directly, see wrapper below
    */
    if (cell == nullptr)
        return;
    sc_likelihood(params_vec, *cell, total_likelihood);

    likelihood_recr(params_vec, cell->daughter1, total_likelihood);
    likelihood_recr(params_vec, cell->daughter2, total_likelihood);
}

double total_likelihood(const std::vector<double> &params_vec, std::vector<double> &grad
    /*
    * total_likelihood of cell tree, to be maximized
    */

    // MOMAdata cell = *(MOMAdata *) c;
    double total_likelihood = 0;

    likelihood_recr(params_vec,  (MOMAdata *) c, total_likelihood);
    return total_likelihood;
```

Minimizer

- nlopt

```
void minimize_wrapper(double (*target_func)(const std::vector<double> &x, std::vector<d
                      MOMAdata &cell,
                      Parameter_set &params,
                      double relative_tol = 1e-10)
```

with connfig file containing :

- bound variable: gamma_lambda
- free variable: var_lambda
- fixed varibale mean_q

```
gamma_lambda = 4, 0.01, -100, 100
var_lambda = 4, 0.001
mean_q = 4
```

Current minimizer: COBYLA

- Constrained Optimization By Linear Approximation (COBYLA)
- Implementation of Powell's method:
  - pick initial x0 and two directions h1, h2
  - starting from x0 1D optimization along first direction h1 -> find x1
  - starting from x1 1D optimization along first direction h2 -> find x2
  - h3 connects x0 and x2
  - starting from x2 1D optimization along first direction h3 -> find x3
  - ...

Parameters

- Growth rate fluctualtions params:
  - mean_lambda;
  - gamma_lambda;
  - var_lambda;
- gfp fluctuation params
  - mean_q;
  - gamma_q;
  - var_q;
  - beta;
- variance guess for length and gfp
  - var_x;
  - var_g;

- cell division:
  - var_dx;
  - var_dg;