



FASTlib: A library for Fundamental Algorithmic and Statistical Tools

Ryan Riegel and Nishant Mehta

`rriegel@cc.gatech.edu`, `niche@cc.gatech.edu`

FASTlab, Georgia Institute of Technology

Outline

- Trouble installing? Ask us after class.
- Starting a project in FASTlib
- Parsing the command line
- Loading a data set from file
- Linear algebra and other math
- Writing linkable code
- Running experiments with FASTexec

Outline

- Trouble installing? Ask us after class.
- Starting a project in FASTlib
- Parsing the command line
- Loading a data set from file
- Linear algebra and other math
- Writing linkable code
- Running experiments with FASTexec

Starting a project

1. `cd` to `contrib/your_name/my_prog/`
2. Create a file called `build.py`
3. Fill it with:

```
binrule(  
    name = "my_prog",  
    sources = ["main.cc"],  
    deplibs = ["fastlib:fastlib"]  
)
```

See also `examples/platonic_allnn/build.py`

Compile with `fl-build my_prog`

Starting a project

4. Create file `main.cc`
5. Fill it with:

```
#include <fastlib/fastlib.h>
```

```
int main(int argc, char *argv[]) {  
    fx_init(argc, argv);  
    fx_done();  
    return 0;  
}
```

Outline

- Trouble installing? Ask us after class.
- Starting a project in FASTlib
- Parsing the command line
- Loading a data set from file
- Linear algebra and other math
- Writing linkable code
- Running experiments with FASTexec

Command line input

- Between `fx_init` and `fx_done`:

```
char *data_file =  
    fx_param_str(NULL, "data", "in.csv");  
printf("%s\n", data_file);
```

- `./my_prog --data=foo` results in `foo`
- `./my_prog` results in `in.csv`

Note: You'll see lots of other output from `fx` (FASTExec)

Command line input

Other handy commands:

- `fx_param_str_req(NULL, "data")`
- `fx_param_int(NULL, "k", 5)`
- `fx_param_double(NULL, "h", 1.0)`
- `fx_param_bool(NULL, "do_naive", 0)`
- `fx_param_exists(NULL, "blah")`

Outline

- Trouble installing? Ask us after class.
- Starting a project in FASTlib
- Parsing the command line
- Loading a data set from file
- Linear algebra and other math
- Writing linkable code
- Running experiments with FASTexec

Reading from file

- After setting `data_file`:

```
Matrix data_mat;  
data::Load(data_file, &data_mat);
```

- Later on, we can use:

```
data::Save(out_file, results_mat);
```

Caveat: Matrices transpose on load/save; column major!

Alternately: Can read `.arff` with class `Dataset`

Outline

- Trouble installing? Ask us after class.
- Starting a project in FASTlib
- Parsing the command line
- Loading a data set from file
- **Linear algebra and other math**
- Writing linkable code
- Running experiments with FASTexec

Ex: Distance matrix

Initialize:

```
int n = data_mat.n_cols();
```

```
Matrix dist_mat;
```

```
dist_mat.Init(n, n);
```

Ex: Distance matrix

```
for (int j = 0; j < n; ++j) {
    Vector j_vec;
    data_mat.MakeColumnVector(j, &j_vec);

    for (int i = 0; i < n; ++i) {
        Vector i_vec;
        data_mat.MakeColumnVector(i, &i_vec);

        double dist_sq =
            la::DistSqEuclidean(i_vec, j_vec);
        dist_mat.set(i, j, sqrt(dist_sq));
    }
}
```

Ex: Distance matrix

```
for (int j = 0; j < n; ++j) {  
    Vector j_vec;  
    data_mat.MakeColumnVector(j, &j_vec);  
  
    for (int i = 0; i < n; ++i) {  
        Vector i_vec;  
        data_mat.MakeColumnVector(i, &i_vec);  
  
        double dist_sq =  
            la::DistSqEuclidean(i_vec, j_vec);  
        dist_mat.set(i, j, sqrt(dist_sq));  
    }  
}
```

Note matrix/vector access.

Ex: Distance matrix

```
for (int j = 0; j < n; ++j) {
    Vector j_vec;
    data_mat.MakeColumnVector(j, &j_vec);

    for (int i = 0; i < n; ++i) {
        Vector i_vec;
        data_mat.MakeColumnVector(i, &i_vec);

        double dist_sq =
            la::DistSqEuclidean(i_vec, j_vec);
        dist_mat.set(i, j, sqrt(dist_sq));
    }
}
```

Note application of BLAS.

Ex: Kernel matrix

```
GaussianKernel gaussian;
gaussian.Init(fx_param_double(NULL, "h", 1.0));

for (int j = 0; j < n; ++j) {
    Vector j_vec;
    data_mat.MakeColumnVector(j, &j_vec);

    for (int i = 0; i < n; ++i) {
        Vector i_vec;
        data_mat.MakeColumnVector(i, &i_vec);

        double dist_sq =
            la::DistSqEuclidean(i_vec, j_vec);
        kernel_mat.set(i, j, gaussian.EvalUnnormOnSq(dist_sq));
    }
}

int dim = data_mat.n_rows();
la::Scale(gaussian.CalcNormConstant(dim), kernel_mat);
```

Ex: Kernel PCA

```
Matrix averaging_mat;  
averaging_mat.Init(n, n);  
averaging_mat.SetAll(1.0 / n);
```

```
Matrix avg_by_kernel, avg_kernel_avg;  
Matrix centered_mat;
```

```
la::MulInit(averaging_mat, kernel_mat, &avg_by_kernel);  
la::SubInit(avg_by_kernel, kernel_mat, &centered_mat);  
la::TransposeSquare(&avg_by_kernel);  
la::SubFrom(avg_by_kernel, &centered_mat);  
la::MulInit(averaging_mat, avg_by_kernel, &avg_kernel_avg);  
la::AddTo(avg_kernel_avg, &centered_mat);
```

```
Vector eigenvalues;  
Matrix eigenvectors;  
la::EigenvectorsInit(centered_mat, &eigenvalues, &eigenvectors);
```

Ex: Kernel PCA

```
Matrix averaging_mat;  
averaging_mat.Init(n, n);  
averaging_mat.SetAll(1.0 / n);
```

```
Matrix avg_by_kernel, avg_kernel_avg;  
Matrix centered_mat;
```

```
la::MulInit(averaging_mat, kernel_mat, &avg_by_kernel);  
la::SubInit(avg_by_kernel, kernel_mat, &centered_mat);  
la::TransposeSquare(&avg_by_kernel);  
la::SubFrom(avg_by_kernel, &centered_mat);  
la::MulInit(averaging_mat, avg_by_kernel, &avg_kernel_avg);  
la::AddTo(avg_kernel_avg, &centered_mat);
```

```
Vector eigenvalues;  
Matrix eigenvectors;  
la::EigenvectorsInit(centered_mat, &eigenvalues, &eigenvectors);
```

Outline

- Trouble installing? Ask us after class.
- Starting a project in FASTlib
- Parsing the command line
- Loading a data set from file
- Linear algebra and other math
- **Writing linkable code**
- Running experiments with FASTexec

Linkable code

1. Separate “algorithmic” portion into its own class.
In `kernel_matrix_algs.h`:

```
class KernelMatrixAlgs {
    Matrix kernel_mat_;

public:
    Init(const Matrix &data_mat, fx_module *module);
    KernelPCA(Vector *eigenvalues, Matrix *eigenvectors);
};
```

2. In `kernel_matrix_algs.cc`, functions for the proceeding algorithms.

Linkable code

3. Change `fx_param_double(NULL, "h", 1.0)` to
`fx_param_double(module , "h", 1.0)`
4. Obtain module in calling function (main) with

```
KernelMatrixAlgs kernel_matrix_algs;  
kernel_matrix_algs.Init(  
    data_mat,  
    fx_submodule(NULL, "algs", "algs"));
```

Now, h is set with:

```
./my_prog --algs/h=0.3
```

Linkable code

5. Add a librulere to the `build.py`:

```
librule(  
    name = "kernel_matrix_algs",  
    headers = ["kernel_matrix_algs.h"],  
    sources = ["kernel_matrix_algs.cc"],  
    deplibs = ["fastlib:fastlib"]  
)  
binrule(  
    name = "my_prog",  
    sources = ["main.cc"],  
    deplibs = [":kernel_matrix_algs"]  
)
```

Outline

- Trouble installing? Ask us after class.
- Starting a project in FASTlib
- Parsing the command line
- Loading a data set from file
- Linear algebra and other math
- Writing linkable code
- **Running experiments with FASTexec**

FASTexec

You can type:

```
./my_prog --data=my_data.csv --algs/h=0.1  
./my_prog --data=my_data.csv --algs/h=0.2  
./my_prog --data=my_data.csv --algs/h=0.3
```

Or you can use:

```
fx-run my_experiment ./my_prog  
--data=../../../../my_data.csv  
--algs/h=0.1,0.2,0.3
```

Creates new working directories

```
fx/my_prog/my_experiment/algs_h_0.1
```