Step by Step guide to add a new fit model function into the GPUFIT

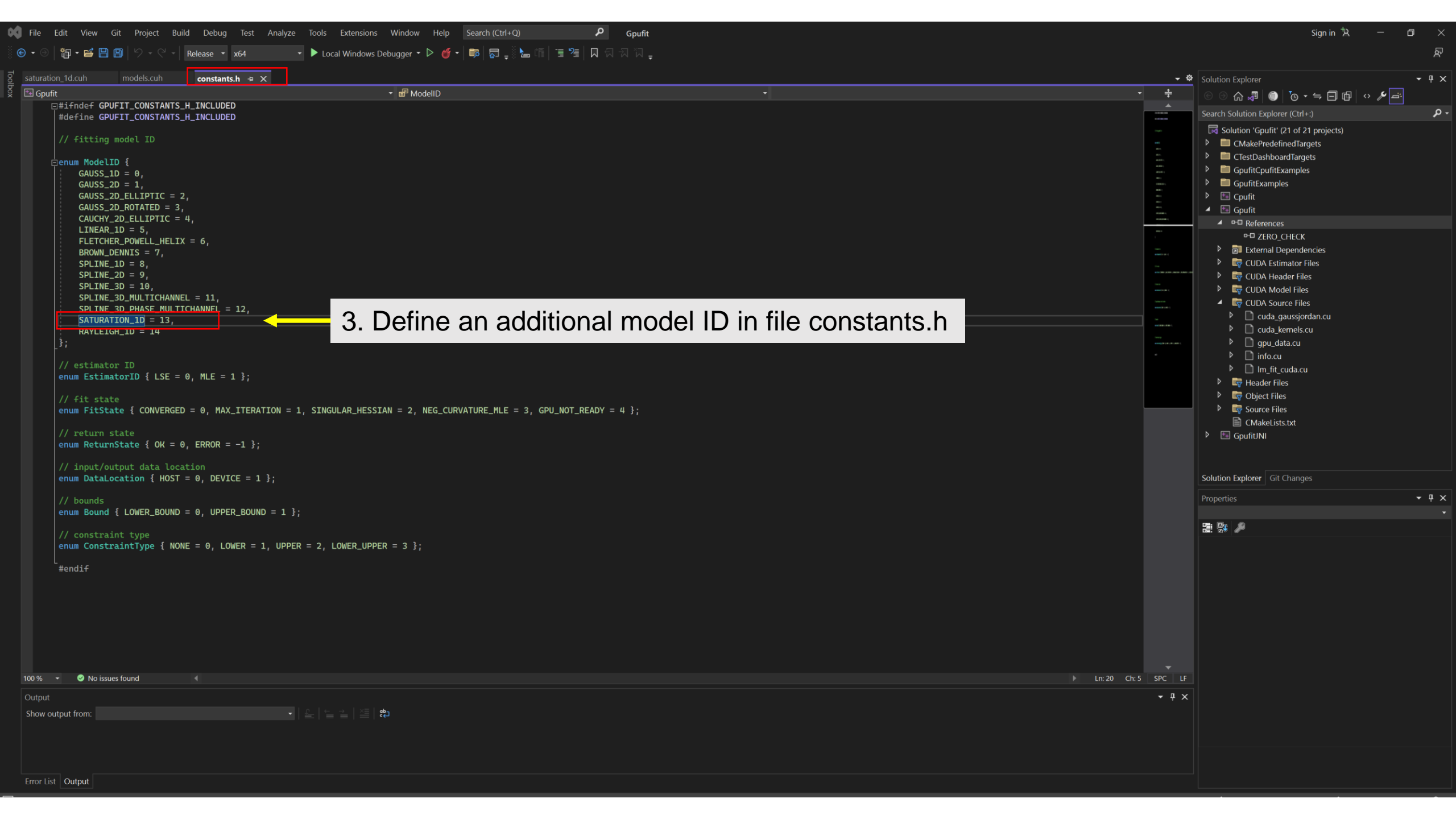GitHub repository: https://github.com/gpufit/Gpufit

Documentation   https://gpufit.readthedocs.io/en/latest/index.html

Steps:

1. Download CUDA

    [CUDA](#) Toolkit 6.5 or later (tested with 6.5-11.4)

2. Download Visual Studio 2022

3. Define an additional model ID in file constants.h

```cpp
#ifndef GPUFIT_CONSTANTS_H_INCLUDED
#define GPUFIT_CONSTANTS_H_INCLUDED

// fitting model ID

enum ModelID {
    GAUSS_1D = 0,
    GAUSS_2D = 1,
    GAUSS_2D_ELLIPTIC = 2,
    GAUSS_2D_ROTATED = 3,
    CAUCHY_2D_ELLIPTIC = 4,
    LINEAR_1D = 5,
    FLETCHER_POWELL_HELIX = 6,
    BROWN_DENNIS = 7,
    SPLINE_1D = 8,
    SPLINE_2D = 9,
    SPLINE_3D = 10,
    SPLINE_3D_MULTICHANNEL = 11,
    SPLINE_3D_PHASE_MULTICHANNEL = 12,
    SATURATION_1D = 13,
    RAYLEIGH_ID = 14
};

// estimator ID
enum EstimatorID { LSE = 0, MLE = 1 };

// fit state
enum FitState { CONVERGED = 0, MAX_ITERATION = 1, SINGULAR_HESSIAN = 2, NEG_CURVATURE_MLE = 3, GPU_NOT_READY = 4 };

// return state
enum ReturnState { OK = 0, ERROR = -1 };

// input/output data location
enum DataLocation { HOST = 0, DEVICE = 1 };

// bounds
enum Bound { LOWER_BOUND = 0, UPPER_BOUND = 1 };

// constraint type
enum ConstraintType { NONE = 0, LOWER = 1, UPPER = 2, LOWER_UPPER = 3 };

#endif
```

4. Implement a CUDA device function within a newly created .cuh file in folder Gpufit/Gpufit/models according to the following template.
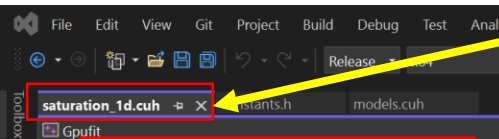

```cuda
#ifndef GPUFIT_SATURATION1D_CUH_INCLUDED
#define GPUFIT_SATURATION1D_CUH_INCLUDED

__device__ void calculate_saturation1d(
    float const * parameters,
    int const n_fits,
    int const n_points,
    float * value,
    float * derivative,
    int const point_index,
    int const fit_index,
    int const chunk_index,
    char * user_info,
    std::size_t const user_info_size)
{
    // indices

    REAL* user_info_float = (REAL*)user_info;
    REAL x = 0;
    if (!user_info_float)
    {
        x = point_index;
    }
    else if (user_info_size / sizeof(REAL) == n_points)
    {
        x = user_info_float[point_index];
    }
    else if (user_info_size / sizeof(REAL) > n_points)
    {
        int const chunk_begin = chunk_index * n_fits * n_points;
        int const fit_begin = fit_index * n_points;
        x = user_info_float[chunk_begin + fit_begin + point_index];
    }

    /////////////////////////// value ///////////////////////////

    value[point_index] =  parameters[0] - parameters[0] * exp(-x/ parameters[1]) ;          // formula calculating fit model values

    /////////////////////////// derivative ///////////////////////////
    float * current_derivative = derivative + point_index;

    current_derivative[0 * n_points] = 1- exp(-x/ parameters[1]) ;  // formula calculating derivative values with respect to parameters[0]
    current_derivative[1 * n_points] = -(parameters[0]* x * exp(-x/ parameters[1]))/ (parameters[1] * parameters[1]);  // formula calculating derivative values with respect to parameters[1]
}

#endif GPUFIT_SATURATION1D_CUH_INCLUDED
```

partial derivatives of the model, with respect to the model parameters
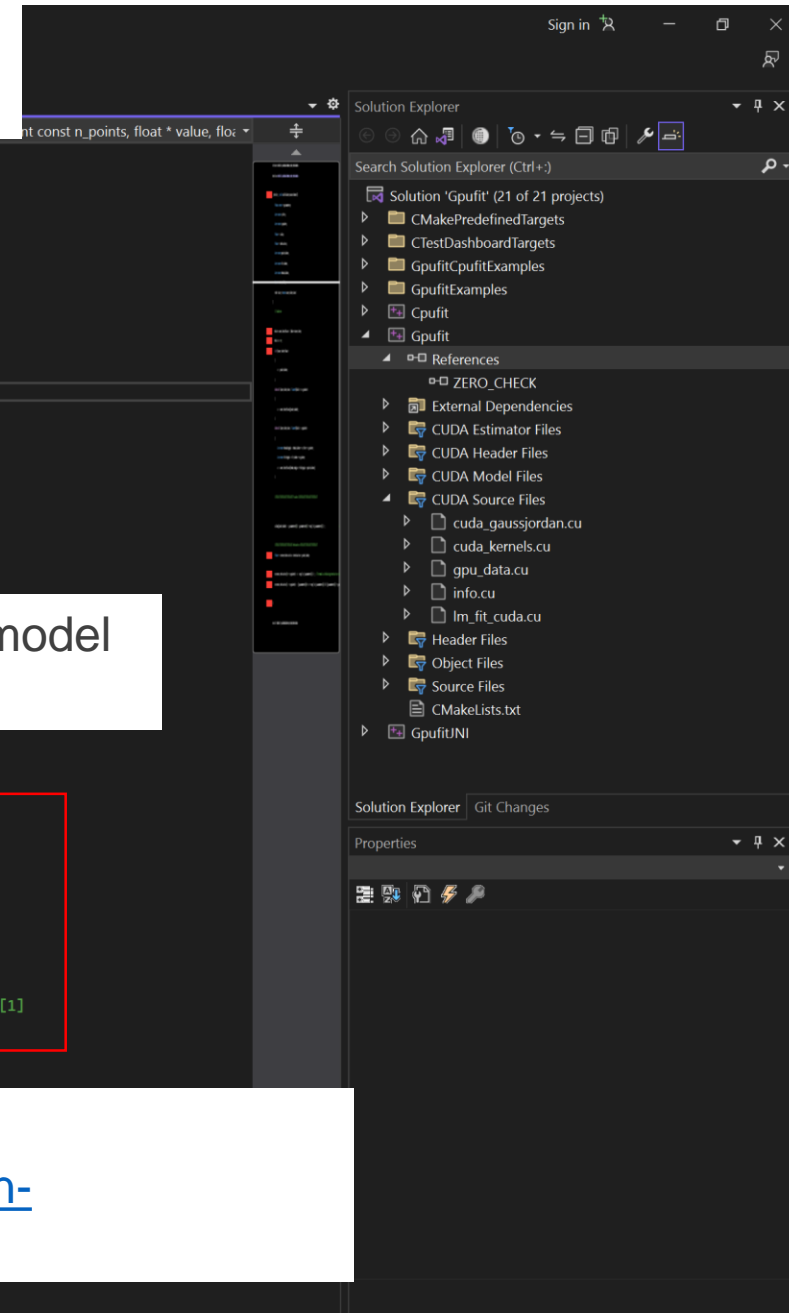
See the examples model functions in thin link:
https://github.com/ComputationalOpticsGroup/Burst-Sparse-Acquisition-Postprocessing/tree/main/GPUFit/Gpufit-1.2.0/Gpufit/models

5. Include the newly created .cuh file in models.cuh

Add a switch case in the CUDA device function calculate_model() in file models.cuh to allow calling the newly added model function.

See the next page (Continued)

```cpp
#ifndef GPUFIT_MODELS_CUH_INCLUDED
#define GPUFIT_MODELS_CUH_INCLUDED

#include <assert.h>
#include "linear_1d.cuh"
#include "gauss_1d.cuh"
#include "gauss_2d.cuh"
#include "gauss_2d_elliptic.cuh"
#include "gauss_2d_rotated.cuh"
#include "cauchy_2d_elliptic.cuh"
#include "fletcher_powell_helix.cuh"
#include "brown_dennis.cuh"
#include "spline_1d.cuh"
#include "spline_2d.cuh"
#include "spline_3d.cuh"
#include "spline_3d_multichannel.cuh"
#include "spline_3d_phase_multichannel.cuh"
#include "saturation_1d.cuh"
#include "rayleigh_1d.cuh"

__device__ void calculate_model(
    ModelID const model_id,
    REAL const * parameters,
    int const n_fits,
    int const n_points,
    REAL * value,
    REAL * derivative,
    int const point_index,
    int const fit_index,
    int const chunk_index,
    char * user_info,
    int const user_info_size)
{
    switch (model_id)
    {
    case GAUSS_1D:
        calculate_gauss1d(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
        break;
    case GAUSS_2D:
        calculate_gauss2d(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
        break;
    case GAUSS_2D_ELLIPTIC:
        calculate_gauss2delliptic(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
        break;
    case GAUSS_2D_ROTATED:
        calculate_gauss2drotated(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
        break;
    case CAUCHY_2D_ELLIPTIC:
        calculate_cauchy2delliptic(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
        break;
    case LINEAR_1D:
```

```
case BROWN_DENNIS:
    calculate_brown_dennis(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
    break;
case SPLINE_1D:
    calculate_spline1d(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
    break;
case SPLINE_2D:
    calculate_spline2d(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
    break;
case SPLINE_3D:
    calculate_spline3d(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
    break;
case SPLINE_3D_MULTICHANNEL:
    calculate_spline3d_multichannel(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
    break;
case SPLINE_3D_PHASE_MULTICHANNEL:
    calculate_spline3d_phase_multichannel(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
    break;
case SATURATION_1D:
    calculate_saturation1d(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
    break;
case RAYLEIGH_1D:
    calculate_rayleigh1d(parameters, n_fits, n_points, value, derivative, point_index, fit_index, chunk_index, user_info, user_info_size);
    break;
default:
    assert(0); // unknown model ID
}
}

void configure_model(ModelID const model_id, int & n_parameters, int & n_dimensions)
{
    switch (model_id)
    {
    case GAUSS_1D:              n_parameters = 4; n_dimensions = 1; break;
    case GAUSS_2D:              n_parameters = 5; n_dimensions = 2; break;
    case GAUSS_2D_ELLIPTIC:     n_parameters = 6; n_dimensions = 2; break;
    case GAUSS_2D_ROTATED:      n_parameters = 7; n_dimensions = 2; break;
    case CAUCHY_2D_ELLIPTIC:    n_parameters = 6; n_dimensions = 2; break;
    case LINEAR_1D:             n_parameters = 2; n_dimensions = 1; break;
    case FLETCHER_POWELL_HELIX: n_parameters = 3; n_dimensions = 1; break;
    case BROWN_DENNIS:          n_parameters = 4; n_dimensions = 1; break;
    case SPLINE_1D:             n_parameters = 3; n_dimensions = 1; break;
    case SPLINE_2D:             n_parameters = 4; n_dimensions = 2; break;
    case SPLINE_3D:             n_parameters = 5; n_dimensions = 3; break;
    case SPLINE_3D_MULTICHANNEL:       n_parameters = 5; n_dimensions = 4; break;
    case SPLINE_3D_PHASE_MULTICHANNEL: n_parameters = 6; n_dimensions = 4; break;
    case SATURATION_1D: n_parameters = 2; n_dimensions = 1; break;
    case RAYLEIGH_1D: n_parameters = 1; n_dimensions = 1; break;
    default: throw std::runtime_error("unknown model ID");
    }
}
```

Add a switch case in function configure_model() in file models.cuh.

6. Re-build the Gpufit project.

Code   Blame   323 lines (262 loc) · 13.9 KB

Raw

```
39    cuda_available_func = lib.gpufit_cuda_available
40    cuda_available_func.restype = c_int
41    cuda_available_func.argtypes = None
42
43    # gpufit_get_cuda_version function in the dll
44    get_cuda_version_func = lib.gpufit_get_cuda_version
45    get_cuda_version_func.restype = c_int
46    get_cuda_version_func.argtypes = [POINTER(c_int), POINTER(c_int)]
47
48
49    class ModelID:
50        GAUSS_1D = 0
51        GAUSS_2D = 1
52        GAUSS_2D_ELLIPTIC = 2
53        GAUSS_2D_ROTATED = 3
54        CAUCHY_2D_ELLIPTIC = 4
55        LINEAR_1D = 5
56        FLETCHER_POWELL = 6
57        BROWN_DENNIS = 7
58        SPLINE_1D = 8
59        SPLINE_2D = 9
60        SPLINE_3D = 10
61        SPLINE_3D_MULTICHANNEL = 11
62        SPLINE_3D_PHASE_MULTICHANNEL = 12
63        SATURATION_1D = 13
64
65
66    class EstimatorID:
67        LSE = 0
68        MLE = 1
69
70
```

Add the model ID in file gpufit.py