API Collection

Quadrature

Approximate the definite integral of a function over a finite or infinite interval.

Overview

Quadrature provides an approximation of the definite integral of a function, over a finite or infinite interval.

Quadrature is a historic term for determining the area under a curve. Often, this was done by breaking the area into smaller shapes, whose area could be easily calculated (such as rectangles), and summing these smaller areas to obtain an approximate result.

In modern terms this process is called *definite integration*. The Accelerate framework's Quadrature functionality provides an approximation of the definite integral of a function, over a finite or infinite interval, performed by evaluating the function at a series of points within the interval.

The Quadrature library provides a Swift-only API, based on the `Quadrature` structure, and both Swift and Objective-C APIs. For the latter, the `quadrature_integrate()` function performs this calculation using any one of three algorithms described in the Integrators section below.

The Integration Callback

Note

To avoid confusion over the word *function*, this document refers to the mathematical function that is to be integrated as the *integrand*.

To represent the integrand, use a C function of the following type, defined in `integration.h`:

```
typedef void (*quadrature_function_array)(void * _Null_unspecified __arg,
                                          size_t __n,
                                          const double *__x,
                                          double *__y
);
```

This function is the *integration callback*. This is a function that processes values in an input array, and produces corresponding result values in an output array; the function should not do anything else, and it must return `void`. The input values are *x* values within the interval over which the integrand is being integrated, and the output values are the corresponding values *y = integrand(x)* at those points.

There are two other parameters: a `void*` pointer `arg`, that you will supply at call time, and a size for the arrays (for details, consult the header file `integration.h`). The pointer `arg` is available in case you need to reference some outside object from inside your callback; however if your callback only needs the `x` and `y` arguments, you can ignore `arg`.

When the integration callback has been defined, you can package it in a struct of type `quadrature_integrate_function`, defined in `integration.h`:

```
quadrature_integrate_function f;
f.fun = integrationCB;
f.fun_arg = myArg;
```

where `integrationCB` is the integration callback and `myArg` is a `void*` pointer value to be passed to `integrationCB` as the first parameter, `arg`. If the integration callback has been written to ignore this value, just pass `NULL`.

The struct `f` can now be passed as the first argument to `quadrature_integrate()`, which will supply the input and output arrays, fill the input array with *x* values, and perform the integration by calling `integrationCB` as many times as necessary.

Integration Options

Options that control the logic of the integration are specified as fields of a struct of type `quadrature_integrate_options`, defined in `integration.h`. The fields are interpreted as follows:

- `integrator`: A constant that identifies one of three available integration algorithms (see the Integrators section below):
- `QUADRATURE_INTEGRATE_QNG` selects the QNG algorithm.

- `QUADRATURE_INTEGRATE_QAG` selects the QAG algorithm.
- `QUADRATURE_INTEGRATE_QAGS` selects the QAGS algorithm.
- `abs_tolerance`: A value of type `double`; requested absolute tolerance on the result.
- `rel_tolerance`: A value of type `double`; requested relative tolerance on the result.
- `qag_points_per_interval`: Number of points per subinterval. Used by the QAG integrator only; other integrators ignore this value. Can be 0, 15, 21, 31, 41, 51, 61. 0 maps to the default 21.
- `max_intervals`: When you do not provide a workspace (see the Providing a Workspace section below), this is the maximum number of subintervals in the subdivision used by QAG and QAGS integrators. `max_intervals` is ignored by QAG and QAGS if you provide a workspace, and always ignored by QNG.

Integrators

Quadrature has three different integrators available to perform the integration, called QNG, QAG, and QAGS. Each integrator is a specific implementation of an algorithm of the same name. The algorithms are all variants of the well-known Gauss-Kronrod method, and their integrators are C ports of the corresponding routines in the QUADPACK library:

- QNG is a simple nonadaptive automatic integrator using Gauss-Kronrod-Patterson quadrature coefficients. It evaluates 21, 43, or 87 points in the interval until the requested accuracy is reached.
- QAG is globally adaptive – it divides the integration interval into a number of subintervals depending on the memory available and/or the `max_intervals` option (see the Providing a Workspace section below). In each interval it uses 0, 15, 21, 31, 41, 51, or 61 points depending on the `qag_points_per_interval` option.
- QAGS is globally adaptive and permits infinite bounds on the integration interval. Within each subinterval it uses 21 points if both bounds are finite, or 15 if one or both bounds are infinite. The algorithm is accelerated by Peter Wynn's epsilon algorithm. If at least one of the interval bounds is infinite, this is equivalent to the QUADPACK QAGI routine. Otherwise, this is equivalent to the QUADPACK QAGS routine.

To select an integrator for a particular integrand, the following decision tree is recommended:

Integration over a finite region

1. If performance is not a concern and you don't know much about the specifics of the problem, use QAGS.
2. Otherwise, if the integrand is smooth, use QNG – or QAG if the requested tolerance couldn't be reached with QNG.

3. Otherwise, if there are discontinuities or singularities of the integrand or of its derivative, and you know where they are, split the integration range at these points and analyze each subinterval.
4. Otherwise, if the integrand has end point singularities, use QAGS.
5. Otherwise, if the integrand has an oscillatory behavior of nonspecific type, and no singularities, use QAG with 61 points per interval.
6. Otherwise, use QAGS.

Integration over an infinite region

1. If the integrand decays rapidly toward zero, truncate the interval and use the finite interval decision tree.
2. Otherwise, if you are not constrained by computer time, and do not wish to analyze the problem further, use QAGS.
3. Otherwise, if the integrand has a non-smooth behavior in the range, and you know where it occurs, split off these regions and use the appropriate finite range routines to integrate over them. Then begin this tree again to handle the remainder of the region.
4. Otherwise, truncation of the interval, or application of a suitable transformation for reducing the problem to a finite range may be possible.

Providing a Workspace

Optionally, you may allocate a memory area as workspace for quadrature. This is not a requirement and you should generally not provide a workspace unless you need precise control over memory usage. The QAG and QAGS algorithms require a workspace and will automatically allocate it if you do not provide one; the size of the automatically allocated workspace will depend on the `max_intervals` option. The QNG algorithm does not require a workspace.

By allocating the workspace yourself, you have control over the amount of memory used; there is a tradeoff between memory usage and performance.

The QAG algorithm requires, for a given number of subintervals, at least that number times `QUADRATURE_INTEGRATE_QAG_WORKSPACE_PER_INTERVAL` bytes in the workspace. If you provide more memory, the algorithm will be able to use more subintervals.

The QAGS algorithm requires, for a given number of subintervals, at least that number times `QUADRATURE_INTEGRATE_QAGS_WORKSPACE_PER_INTERVAL` bytes in the workspace. If you provide more memory, the algorithm will be able to use more subintervals.

Topics

Quadrature

`struct Quadrature`

A structure that approximates the definite integral of a function over a finite interval.

`quadrature_integrate`

Computes an approximation to the definite integral of a function on a specified interval.

`struct quadrature_integrator`

Constants that specify integration algorithms.

`struct quadrature_status`

Constants that indicate the status of a quadrature operation.