Adaptive numerical Lebesgue integration by set measure estimates

Anton Antonov
MathematicaForPrediction blog at WordPress
MathematicaForPrediction project at GitHub
June, 2016

Introduction

In this document are given outlines and examples of several related implementations of Lebesgue integration, [1], within the framework of `NIntegrate`, [7]. The focus is on the implementations of Lebesgue integration algorithms that have multiple options and can be easily extended (in order to do further research, optimization, etc.) In terms of `NIntegrate`'s framework terminology it is shown how to implement an *integration strategy* or *integration rule* based on the theory of the Lebesgue integral. The full implementation of those strategy and rules -- `LebesgueIntegration`, `LebesgueIntegrationRule`, and `GridLebesgueIntegrationRule` -- are given in the *Mathematica* package [5].

The advantage of using `NIntegrate`'s framework is that a host of supporting algorithms can be employed for preprocessing, execution, experimentation, and testing (correctness, comparison, and profiling.)

Here is a brief description of the integration strategy `LebesgueIntegration` in [5]:
1. prepare a function that calculates measure estimates based on random points or low discrepancy sequences of points in the integration domain;
2. use `NIntegrate` for the computation of one dimensional integrals for that measure estimate function over the range of the integrand function values.

The strategy is adaptive because of the second step -- `NIntegrate` uses (in general) adaptive integration algorithms.

Instead of using an integration strategy we can "tuck in" the whole Lebesgue integration process into an integration rule, and then use that integration rule with the adaptive integration algorithms `NIntegrate` already has. This is done with the implementations of the integration rules `LebesgueIntegrationRule` and `GridLebesgueIntegrationRule`.

Brief theory

Lebesgue integration extends the definition of integral to a much larger class of functions than the class of Riemann integrable functions. The Riemann integral is constructed by partitioning the integrand's domain (on the x axis). The Lebesgue integral is constructed by partitioning the integrand's co-domain (on the y axis). For each value y in the co-domain, find the measure $\mu(y)$ of the corresponding set of points $f^{-1}(y)$ in the domain. Roughly speaking, the Lebesgue integral is then the sum of all the products

$y\,\mu(y)$; see [1]. For our implementation purposes μ is defined differently, and in the rest of this section we follow [3].

Consider the non-negative bound-able measurable function f:

$$y = f(x),\ f(x) \geq 0,\ \ x \in \Omega, \tag{1}$$

We denote by $\mu(y)$ the measure for the points in Ω for which $f(x) \geq y$, i.e.

$$\mu(y) := \ |\{x : x \in \Omega \bigwedge f(x) \geq y\}\ |\ . \tag{2}$$

The Lebesgue integral of $f(x)$ over Ω can be defined as:

$$\int_{\Omega} f(x)\, dl\, x = y_0\, \mu(y_0) + \lim_{n\to\infty,\max(y_i - y_{i-1})\to 0} \sum_{i=1}^{n} \mu(y_i)\,(y_i - y_{i-1}). \tag{3}$$

Further, we can write (1) as

$$\int_{\Omega} f(x)\, dl\, x = y_0\, \mu(y_0) + \int_{y_0}^{y_n} \mu(y)\, dl\, y. \tag{4}$$

The restriction $f(x) \geq 0$ can be handled by defining the following functions f_1 and f_2:

$$f_1(x) := \frac{1}{2}(\,|\,f(x)\,|\, + f(x)), \tag{5}$$

$$f_2(x) := \frac{1}{2}(\,|\,f(x)\,|\, - f(x)), \tag{6}$$

and using the formula

$$\int_{\Omega} f(x)\, dl\, x = \int_{\Omega} f_1(x)\, dl\, x - \int_{\Omega} f_2(x)\, dl\, x. \tag{7}$$

Since finding analytical expressions of μ is hard we are going to look into ways of approximating μ.

For more details see [1,2,3,4].

(Note, that the theoretical outline the algorithms considered can be seen as algorithms that reduce multidimensional integration to one dimensional integration.)

Algorithm walk through

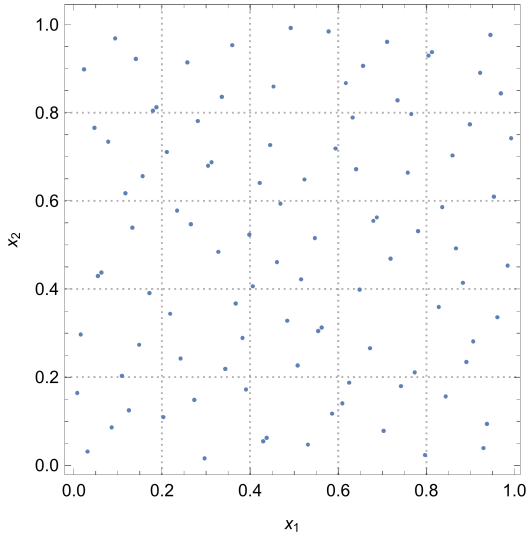
We can see that because of Equation (4) we mostly have to focus on estimating the measure function μ. This section provides a walk through with visual examples of a couple of stochastic ways to do that.

Consider the integral

$$\int_0^1 \int_0^1 \sqrt{1 + x_1 + x_2}\ \ dl\, x_1\, dl\, x_2. \tag{8}$$

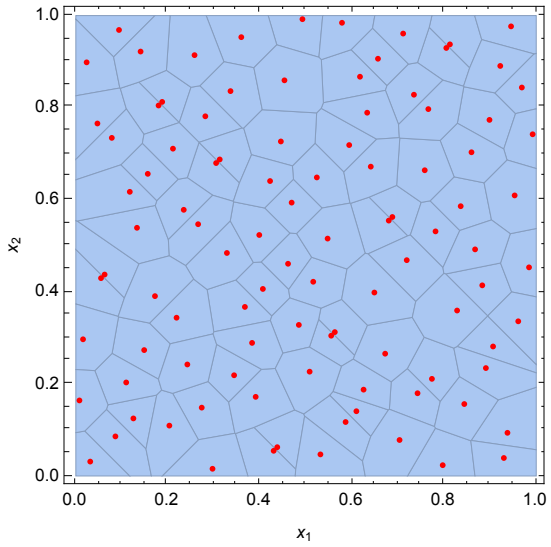
In order to estimate μ in $\Omega := [0,\ 1] \times [0,\ 1]$ for $f(x_1,\ x_2) := \sqrt{1 + x_1 + x_2}$ we are going to generate in Ω a set P of low discrepancy sequence of points, [2]. Here this is done with 100 points of the so called "Sobol" sequence:

```
n = 100;
SeedRandom[0, Method → {"MKL", Method → {"Sobol", "Dimension" → 2}}];
points = RandomReal[{0, 1}, {n, 2}];
ListPlot[points, AspectRatio → Automatic,
 PlotTheme → "Detailed", FrameLabel → {"x₁", "x₂"}]
```



To each point p_i let us assign a corresponding "volume" v_i that can be used to approximate μ with Equation (2). We can of course easily assign such volumes to be $1/|P|$, but as it can be seen on the plot this would be a good approximation for a larger number of points. Here is an example of a different volume assignment using a Voronoi diagram, [10]:

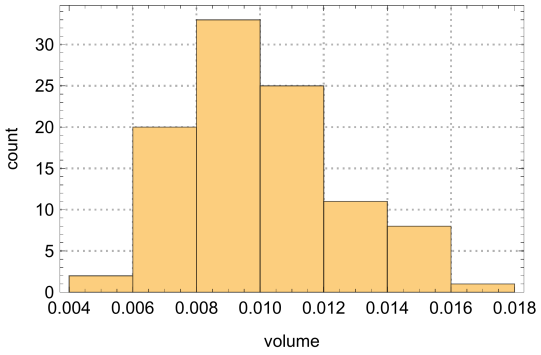
```
vmesh = VoronoiMesh[points, {{0, 1}, {0, 1}}, Frame → True];
Show[{vmesh, Graphics[{Red, Point[points]}]}, FrameLabel → {"x₁", "x₂"}]
```



(The Voronoi diagram for P finds for each point p_i the set of domain points closest to p_i than any other point of P.)

Here is a breakdown of the Voronoi diagram volumes corresponding to the generated points (compare with $0.01 = 1/|P|$) :

```
volumes = PropertyValue[{vmesh, Dimensions[points][[2]]}, MeshCellMeasure];
Histogram[volumes, PlotRange → All,
 PlotTheme -> "Detailed", FrameLabel → {"volume", "count"}]
```



Let us define a function that computes μ according to Equation (2) with the generated points and assigned volumes:

```
EstimateMeasure[fval_?NumericQ, pointVals_, pointVolumes_] :=
  Block[{pinds},
   pinds = Clip[Sign[pointVals - fval], {0, 1}, {0, 1}];
   pointVolumes.pinds
  ];
```

Here is an example call of that function using the Voronoi diagram volumes:

```
EstimateMeasure[1.6, √(2 + Total[#]) & /@ points, volumes]
```

```
0.845833
```

And here is another call using uniform volumes:

```
EstimateMeasure[1.6, √(2 + Total[#]) & /@ points,
  Table[1 / Length[points], {Length[points]}]] // N
```

```
0.85
```

The results can be verified using `ImplicitRegion`:

```
RegionMeasure[ImplicitRegion[√(2 + x1 + x2) ≥ 1.6, {{x1, 0, 1}, {x2, 0, 1}}]]
```

```
0.8432
```

Or using `Integrate`:

```
Integrate[Piecewise[{{1, √(2 + x1 + x2) ≥ 1.6}}, 0], {x1, 0, 1}, {x2, 0, 1}]
```

```
0.8432
```

At this point we are ready to compute the integral estimate using Formula (4) :

```
fvals = √(2 + Total[#]) & /@ points;
Min[fvals] * EstimateMeasure[0, fvals, volumes] +
 NIntegrate[EstimateMeasure[y, fvals, volumes], {y, Min[fvals], Max[fvals]}]
```

```
1.72724
```

To simplify the code we use the symbol `fvals` to hold the values $f(P)$. Note that instead of the true min and max values of f we use estimates of them with `fvals`.

Here is the verification of the result:

```
Integrate[Sqrt[2 + x1 + x2], {x1, 0, 1}, {x2, 0, 1}]
% // N
```

$$\frac{8}{15}\left(16 + 2\sqrt{2} - 9\sqrt{3}\right)$$

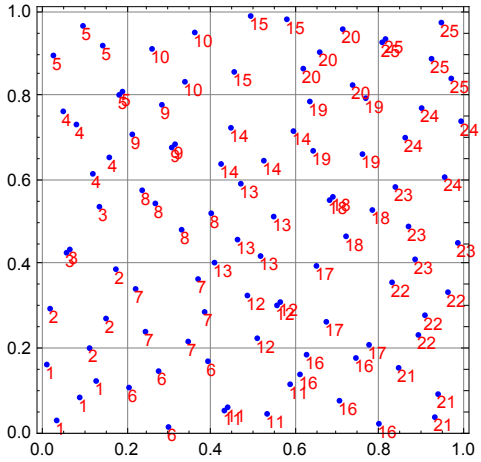
```
1.72798
```

In order to implement the outlined algorithm so it will be more universal we have to consider volumes rescaling, function positivity, and Voronoi diagram implementation(s). For details how these considerations are resolved see the code of the strategy `LebesgueIntegration` in [5].

Further algorithm elaborations

Measure estimate with regular grid cells

The article [3] and book [4] suggest the measure estimation to be done through membership of regular grid of cells. For example, the 100 points generated in the previous section can be grouped by a 5×5 grid:



The following steps describe in detail an algorithm based on the proposed in [3,4] measure estimation method. The algorithm is implemented in [5] for the symbol, `GridLebesgueIntegrationRule`.

1. Generate points filling the $[0, 1]^d$ where d is the dimension of Ω.

2. Partition $[0, 1]^d$ with a regular grid according to specifications.
2.1. Assume the cells are indexed with the integers $I_c \subset \mathbb{N}$, $|I_c| = n$.
2.2. Assume that all cells have the same volume v_c below.

3. For each point find to which cell of the regular grid it belongs to.

4. For each cell have a list of indices corresponding to the points that belong to it.

5. For a given sub-region of integration r rescale to the points to lie within r; denote those points with P_r.
5.1. Compute the rescaling factor for the integration rule; denote with γ.

6. For a given integrand function f evaluate f over all points P_r.

7. For each cell $i \in I_c$ find the min and max values of f.
7.1. Denote with $f_i^{\min}$ and $f_i^{\max}$ correspondingly.

8. For a given value $y_k = f(x_k)$, where k is some integer enumerating the 1D integration rule sampling points, find the coefficients ξ_i, $i \in I_c$ using the following formula:

$$\xi_i = \text{Piecewise}\left[\left\{\left\{1,\ y_k \le f_i^{\min}\right\},\ \left\{0,\ f_i^{\max} < y_k\right\},\ \left\{\frac{\text{Abs}\left[f_i^{\max} - y_k\right]}{\text{Abs}\left[f_i^{\max} - f_i^{\min}\right]},\ f_i^{\min} < y_k < f_i^{\max}\right\}\right\}\right] \tag{9}$$

9. Find the measure estimate of $\mu(f(x) > f(x_k))$ with

$$\gamma \sum_{i=1}^{n} v_c\, \xi_i \tag{10}$$

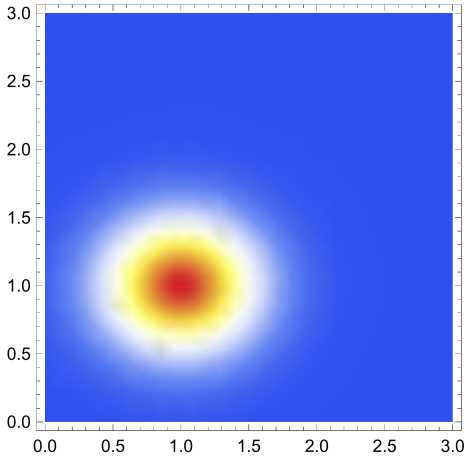
Axis splitting in Lebesgue integration rules

The implementations of Lebesgue integration rules are required to provide a splitting axis for use of the adaptive algorithms. Of course we can assign a random splitting axis, but that might lead often to slower computations. One way to provide splitting axis is to choose the axis that minimizes the sum of the variances of sub-divided regions estimated by samples of the rule points. This is the same approach taken in `NIntegrate`s rule "MonteCarloRule"; for theoretical details see the chapter "7.8 Adaptive and Recursive Monte Carlo Methods" of [11].

In [5] this splitting axis choice is implemented based on integrand function values. Another approach, more in the spirit of the Lebesgue integration, is to select the splitting axis based on variances of the measure function estimates.

Consider the function:

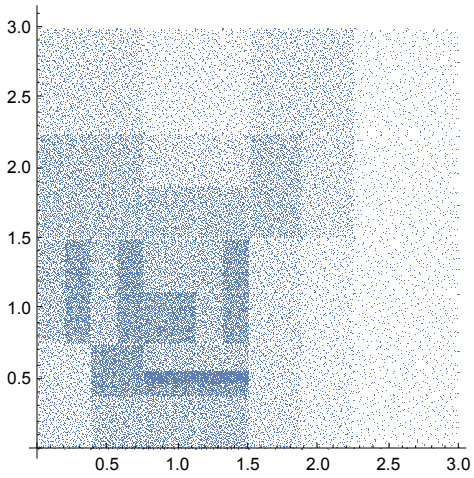
```
DensityPlot[Exp[-3 (x - 1)^2 - 4 (y - 1) ^ 2], {x, 0, 3},
 {y, 0, 3}, PlotRange -> All, ColorFunction -> "TemperatureMap"]
```



Here is an example of sampling points traces using "minimum variance" axis splitting in `LebesgueIntegrationRule`:

```
res = Reap@NIntegrate[Exp[-3 (x - 1)^2 - 4 (y - 1)^2],
    {x, 0, 3}, {y, 0, 3}, Method -> {"GlobalAdaptive",
      Method → {LebesgueIntegrationRule, "Points" → 600, "PointGenerator" → "Sobol",
        "AxisSelector" → "MinVariance"}, "SingularityHandler" → None},
    EvaluationMonitor :→ Sow[{x, y}], PrecisionGoal → 2.5, MaxRecursion → 3];
res[[1]]
ListPlot[res[[2, 1]], AspectRatio → Automatic]
```

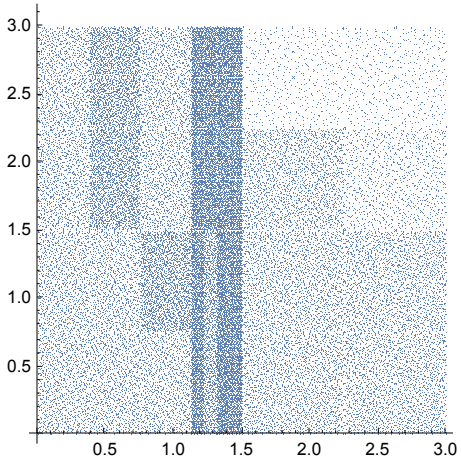
```
0.890916
```



And here are the sampling points with random selection of a splitting axis:

```
res = Reap@NIntegrate[Exp[-3 (x - 1)^2 - 4 (y - 1)^2],
    {x, 0, 3}, {y, 0, 3}, Method -> {"GlobalAdaptive",
      Method → {LebesgueIntegrationRule, "Points" → 600, "PointGenerator" → "Sobol",
        "AxisSelector" → Random}, "SingularityHandler" → None},
    EvaluationMonitor :→ Sow[{x, y}], PrecisionGoal → 2.5, MaxRecursion → 3];
res[[1]]
ListPlot[res[[2, 1]], AspectRatio → Automatic]
```

```
0.892499
```



Here is a more precise estimate of that integral:

```
NIntegrate[Exp[-3 (x - 1)^2 - 4 (y - 1) ^ 2], {x, 0, 3}, {y, 0, 3}]
```

```
0.898306
```

Implementation design within NIntegrate's framework

Basic usages

The strategy and rule implementations in [5] can be used in the following ways.

```
NIntegrate[Sqrt[x], {x, 0, 2}, Method → LebesgueIntegration]
```

```
1.88589
```

```
NIntegrate[Sqrt[x], {x, 0, 2},
 Method → {LebesgueIntegration, Method → "LocalAdaptive",
    "Points" → 2000, "PointGenerator" → "Sobol"}, PrecisionGoal → 3]
```

```
1.88597
```

```
NIntegrate[Sqrt[x], {x, 0, 2},
 Method → {LebesgueIntegrationRule, "Points" → 2000, "PointGenerator" → "Sobol",
    "PointwiseMeasure" → "VoronoiMesh"}, PrecisionGoal → 3]
```

```
1.88597
```

```
NIntegrate[Sqrt[x + y + x], {x, 0, 2}, {y, 0, 3}, {z, 0, 4},
 Method → {GridLebesgueIntegrationRule, Method → "GaussKronrodRule",
    "Points" → 2000, "GridSizes" → 5, "PointGenerator" → "Sobol"}, PrecisionGoal → 3]
```

```
43.6364
```

Options

Here are the options for the implemented strategy and rules in [5]:

LebesgueIntegration	LebesgueIntegrationRule	GridLebesgueIntegrationRule
Method → Automatic PointGenerator → Sobol PointwiseMeasure → Automatic Points → Automatic LebesgueIntegralVariableSymbol → Automatic SymbolicProcessing → 5	Method → ClenshawCurtisRule PointGenerator → Sobol PointwiseMeasure → Automatic Points → Automatic AxisSelector → Automatic	Method → ClenshawCurtisRule PointGenerator → Sobol GridSizes → Automatic Points → Automatic AxisSelector → Automatic

Using variable ranges

Integration with variable ranges works "out of the box."

```
NIntegrate[Sqrt[x + y], {x, 0, 2}, {y, 0, x}]
```

```
2.75817
```

```
NIntegrate[Sqrt[x + y], {x, 0, 2}, {y, 0, x},
 Method → LebesgueIntegration, PrecisionGoal → 2]
```

```
2.75709
```

```
NIntegrate[Sqrt[x + y], {x, 0, 2}, {y, 0, x},
 Method → LebesgueIntegrationRule, PrecisionGoal → 2]
```

```
2.75663
```

Infinite ranges

In order to get correct results with infinite ranges the wrapper

```
Method->{"UnitCubeRescaling","FunctionalRangesOnly"->False, _}
```

has to be used. Here is an example:

```
NIntegrate[1 / (x^2 + 12), {x, 0, ∞}]
```

```
0.45345
```

```
NIntegrate[1 / (x^2 + 12), {x, 0, ∞},
 Method -> {"UnitCubeRescaling", "FunctionalRangesOnly" -> False,
   Method → {LebesgueIntegrationRule, "Points" → 2000}}, PrecisionGoal → 3]
```

```
0.453366
```

For some integrands we have to specify inter-range points or larger `MinRecursion`.

```
NIntegrate[1 / (x^2), {x, 1, ∞}]
```

```
1.
```

```
NIntegrate[1 / (x^2), {x, 1, 12, ∞},
 Method -> {"UnitCubeRescaling", "FunctionalRangesOnly" -> False,
   Method → {LebesgueIntegrationRule, "Points" → 1000}}]
```

```
0.999466
```

```
NIntegrate[1 / (x^2), {x, 1, ∞},
 Method -> {"UnitCubeRescaling", "FunctionalRangesOnly" -> False,
   Method → {LebesgueIntegrationRule, "Points" → 1000}}]
```

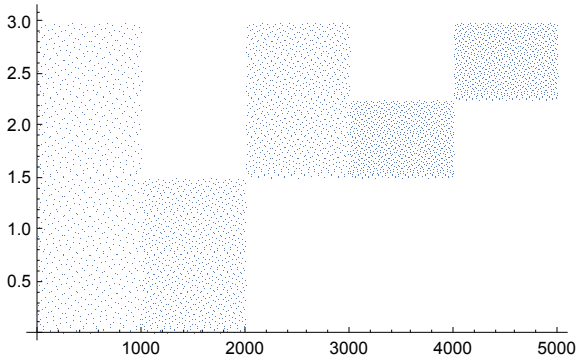
```
0.
```

Evaluation monitoring

With the option “EvaluationMonitor” we can see the sampling points for the strategy and the rules.

This is straightforward for the rules:

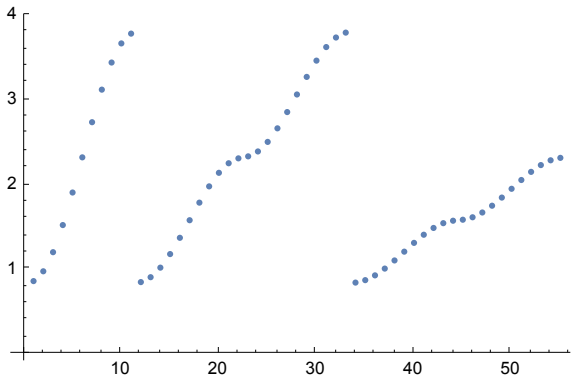
```
res = Reap@NIntegrate[Exp[-3 (x - 1)^2], {x, 0, 3}, Method → LebesgueIntegrationRule,
    EvaluationMonitor :→ Sow[x], PrecisionGoal → 3];
ListPlot[
 res[[
  2,
  1]]]
```



The strategy `LebesgueIntegration` uses an internal variable for the calculation of the Lebesgue integral. In “EvaluationMonitor” either that variable has to be used, or a symbol name has to be passed though the option “LebesgueIntegralVariableSymbol”. Here is an example:

```
res = Reap@NIntegrate[Sqrt[x + y + z], {x, -1, 2},
    {y, 0, 1}, {z, 1, 12}, Method → {LebesgueIntegration, "Points" → 600,
      "PointGenerator" → "Sobol", "LebesgueIntegralVariableSymbol" → fval},
    EvaluationMonitor :→ {Sow[fval]}, PrecisionGoal → 3];
res = DeleteCases[res, fval, ∞];
res[[1]]
ListPlot[res[[2, 1]]]
```

```
87.7618
```



Profiling

We can use `NIntegrate`s utility functions for visualization and profiling in order to do comparison of the implemented algorithms with related ones (like “AdaptiveMonteCarlo”) which `NIntegrate` has (or are plugged-in).

```
Needs["Integration`NIntegrateUtilities`"]
```

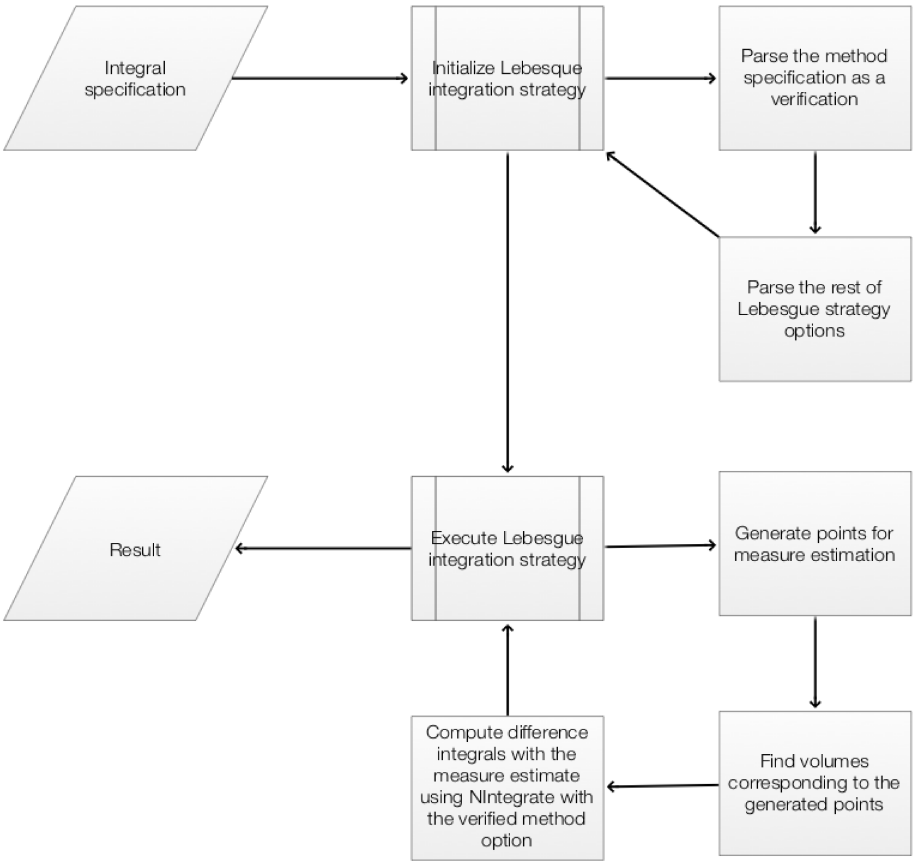
```
fexpr = 1 / (375 / 100 - Cos[x] - Cos[y]);
ranges = {{x, 0, π}, {y, 0, π}};

pgen = "Random";
npoints = 1000;

NIntegrateProfile[
 NIntegrate[fexpr, Evaluate[Sequence @@ ranges], Method →
   {"AdaptiveMonteCarlo", Method → {"MonteCarloRule", "PointGenerator" → pgen,
      "Points" → npoints, "AxisSelector" → "MinVariance"}}, PrecisionGoal → 3]
]
{IntegralEstimate → 2.8527356472097902, Evaluations → 17 000, Timing → 0.0192079}

NIntegrateProfile[
 NIntegrate[fexpr, Evaluate[Sequence @@ ranges], Method → {"GlobalAdaptive",
    Method → {LebesgueIntegrationRule, "PointGenerator" → pgen, "Points" → npoints,
      "AxisSelector" → "MinVariance", Method → "ClenshawCurtisRule"},
    "SingularityHandler" → None}, PrecisionGoal → 3]
]
{IntegralEstimate → 2.836659588960318, Evaluations → 13 000, Timing → 0.384246}
```

Design diagrams

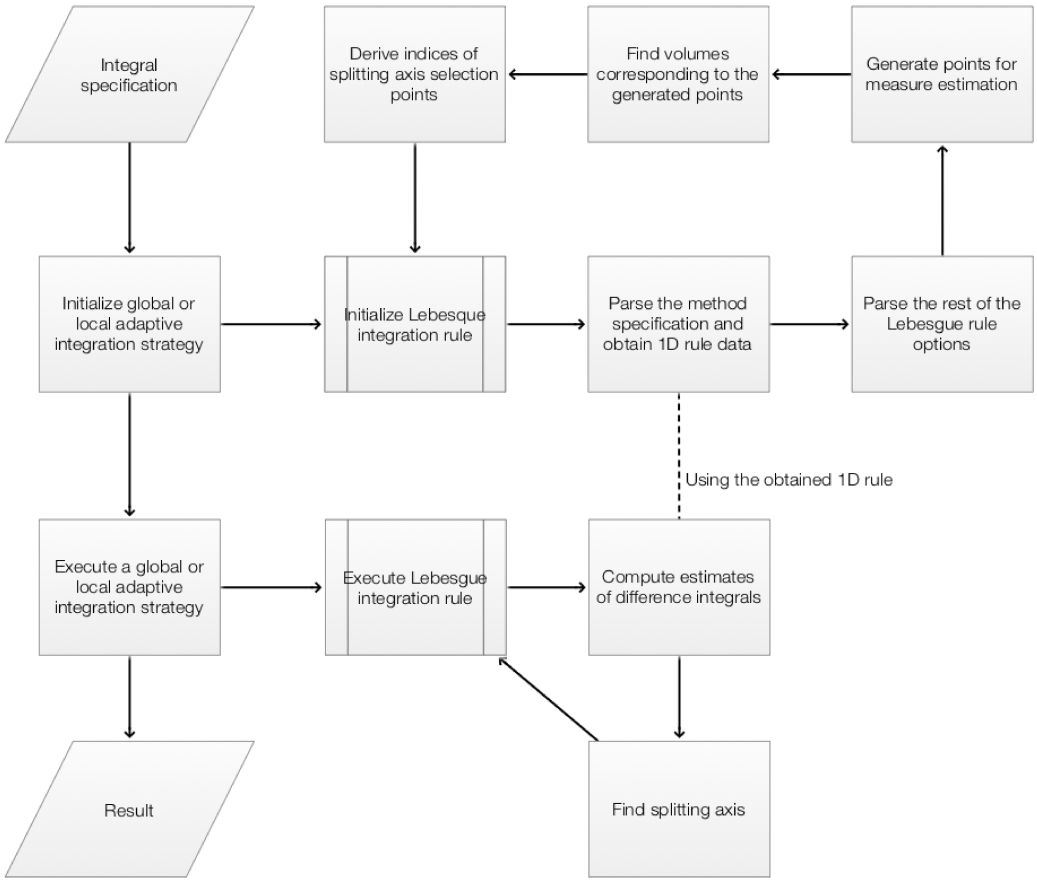
The flow charts below show that the plug-in designs have common elements. In order to make the computations more effective the rule initialization prepares the data that is used in all rule invocations. For the strategy the initialization can be much lighter since the strategy algorithm is executed only once.

In the flow charts the double line boxes designate sub-routines. We can see that so called Hollywood principle “don’t call us, we’ll call you” in Object-oriented programming is manifested.

The following flow chart shows the steps of `NIntegrate`'s execution when the integration strategy `LebesgueIntegration` is used.

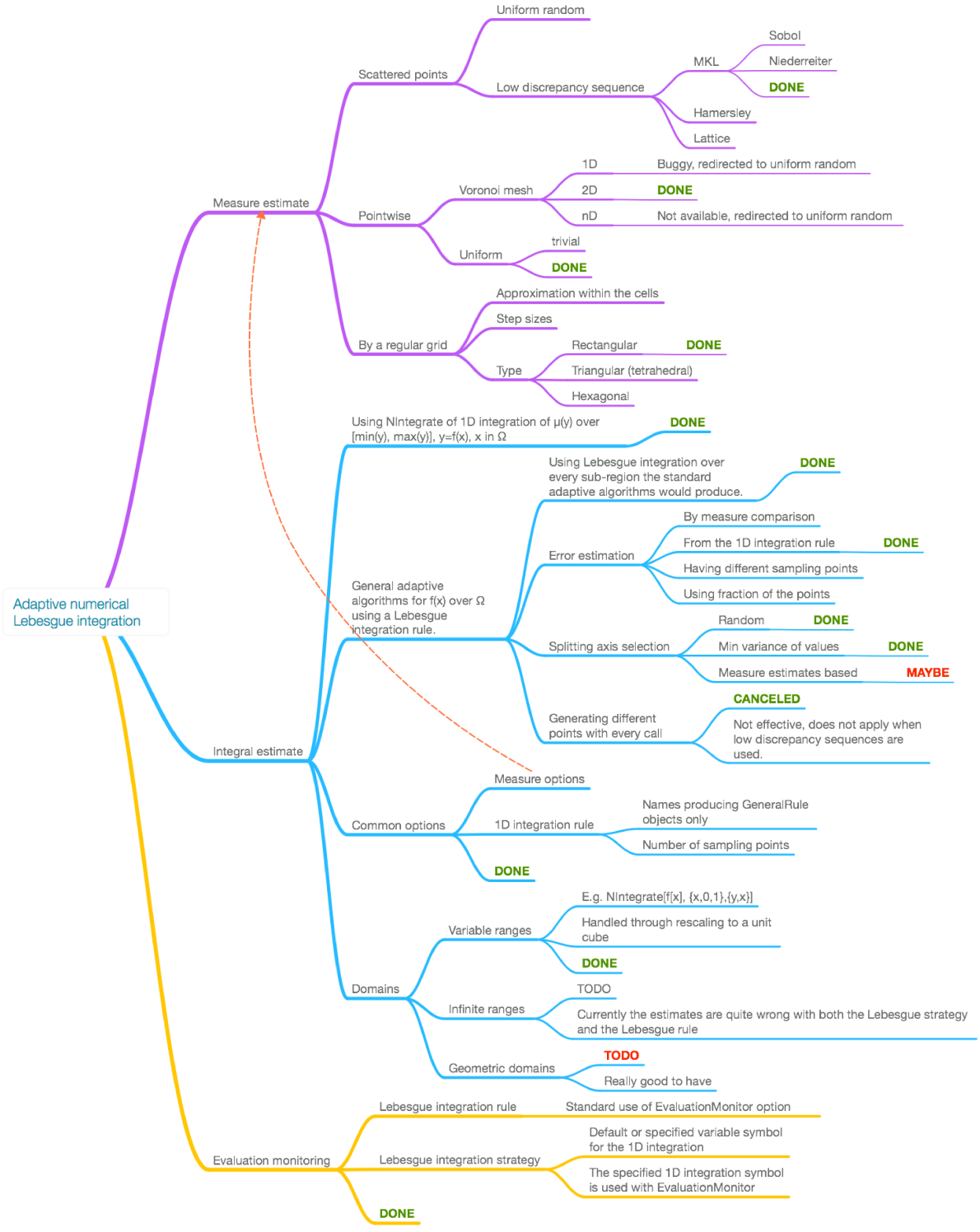


The following flow chart shows the steps of `NIntegrate`'s execution when the integration rule `LebesgueIntegrationRule` is used.



Algorithms versions and options

There are multiple architectural, coding, and interface decisions to make while doing implementations like the ones in [5] and described in this document. The following mind map provides an overview of alternatives and interactions between components and parameters.


Adaptive numerical Lebesgue integration
Measure estimate
Scattered points
Uniform random
Low discrepancy sequence
MKL
Sobol
Niederreiter
DONE
Hamersley
Lattice
Pointwise
Voronoi mesh
1D
Buggy, redirected to uniform random
2D
DONE
nD
Not available, redirected to uniform random
Uniform
trivial
DONE
By a regular grid
Approximation within the cells
Step sizes
Type
Rectangular
DONE
Triangular (tetrahedral)
Hexagonal
Integral estimate
Using NIntegrate of 1D integration of μ(y) over [min(y), max(y)], y=f(x), x in Ω
DONE
General adaptive algorithms for f(x) over Ω using a Lebesgue integration rule.
Using Lebesgue integration over every sub-region the standard adaptive algorithms would produce.
DONE
Error estimation
By measure comparison
From the 1D integration rule
DONE
Having different sampling points
Using fraction of the points
Splitting axis selection
Random
DONE
Min variance of values
DONE
Measure estimates based
MAYBE
Generating different points with every call
CANCELED
Not effective, does not apply when low discrepancy sequences are used.
Common options
Measure options
1D integration rule
Names producing GeneralRule objects only
Number of sampling points
DONE
Domains
Variable ranges
E.g. NIntegrate[f[x], {x,0,1},{y,x}]
Handled through rescaling to a unit cube
DONE
Infinite ranges
TODO
Currently the estimates are quite wrong with both the Lebesgue strategy and the Lebesgue rule
Geometric domains
TODO
Really good to have
Evaluation monitoring
Lebesgue integration rule
Standard use of EvaluationMonitor option
Lebesgue integration strategy
Default or specified variable symbol for the 1D integration
The specified 1D integration symbol is used with EvaluationMonitor
DONE

Alternative implementations

In many ways using the Lebesgue integration rule with the adaptive algorithms is similar to using `NIntegrate`s "AdaptiveMonteCarlo" and its rule "MonteCarloRule". Although it is natural to consider plugging-in the Lebesgue integration rules into "AdaptiveMonteCarlo" at this point `NIntegrate`'s framework does not allow "AdaptiveMonteCarlo" the use of a rule different than "MonteCarloRule".

We can consider using Monte Carlo algorithms for estimating the measures corresponding to a vector of values (that come from a 1D integration rule). This can be easily done, but it is not that effective because of the way `NIntegrate` handles vector integrands and because of stopping criteria issues when the measures are close to 0.

Future plans

One of the most interesting extensions of the described Lebesgue integration algorithms and implementation designs is their extension with more advanced features of *Mathematica* for geometric computation. (Like the functions `VoronoiMesh`, `RegionMeasure`, and `ImplicitRegion` used above.)

Another interesting direction is the derivation and use of symbolic expressions for the measure functions. (Hybrid symbolic and numerical algorithms can be obtained as `NIntegrate`s handling of piecewise functions or the strategy combining symbolic and numerical integration described in [9].)

References

[1] Wikipedia entry, Lebesgue integration, URL: http s://en.wikipedia.org/wiki/Lebesgue_integration .

[2] Wikipedia entry, Low-discrepancy sequence, URL: https://en.wikipedia.org/wiki/Low-discrepancy_sequence .

[3] B. L. Burrows, A new approach to numerical integration, 1. Inst. Math. Applics., 26(1980), 151-173.

[4] T. He, Dimensionality Reducing Expansion of Multivariate Integration, 2001, Birkhauser Boston. ISBN-13:978-1-4612-7414-8 .

[5] A. Antonov, Adaptive Numerical Lebesgue Integration Mathematica Package, 2016, Mathematica-ForPrediction project at GitHub.

[6] A. Antonov, Lebesgue integration, Wolfram Demonstrations Project, 2007. URL: http://demonstrations.wolfram.com/LebesgueIntegration .

[7] "Advanced Numerical Integration in the Wolfram Language", Wolfram Research Inc. URL: https://reference.wolfram.com/language/tutorial/NIntegrateOverview.html .

[8] A. Antonov, "How to implement custom integration rules for use by NIntegrate?", (2016) Mathematica StackExchange answer, URL: http://mathematica.stackexchange.com/a/118326/34008 .

[9] A. Antonov, "How to implement custom NIntegrate integration strategies?", (2016) Mathematica StackExchange answer, URL: http://mathematica.stackexchange.com/a/118922/34008 .

[10] Wikipedia entry, Voronoi diagram, URL: https://en.wikipedia.org/wiki/Voronoi_diagram .

[11] Press, W.H. et al., Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing, 1992, Cambridge University Press, New York, NY, USA.