Numerical Integration Methods Summary

Aleksejs Fomins

Below is presented a short summary of integration method types known to us:

Gaussian Quadrature: The method available in DUNE.

- This method calculates the integral as a linear product of the integrand $f(\vec{r})$ values at specific precomputed points $\vec{r}_i$ with specific precomputed weights w_i, namely $I = \sum_i w_i f(\vec{r}_i)$. Thus the main advantage of this method is its computational cost, which is small for low order polynomial integrands.

- Optimal quadrature points are only available for small dimensions. Finding such point sets for high dimension polynomials is very involved and is known to suffer from finite precision of floating point arithmetic. Alternatively, a suboptimal point distribution can be obtained from a tensor product space of 1D point distributions, whose size grows exponentially with integration dimension.

- Gaussian Quadrature is constructed with the idea of calculating exact integrals for integrands being polynomial up to a given order. However, when integrating over a curved boundary, thte integration elemen is a square root of a polynomial, and polynomials really badly approximate square root, especially for small arguments, which can easily happen for highly curved elements. Not to mention that one, in principle, could with to integrate arbitrary (within reason) functions over the element. Thus

    - Can GQ estimate integration error for non-polynomial functions?

    - Can it be made hierarchical to have control over error by refinement?

    - What would be the convergence rate to compare with other methods?

Interpolatory adaptive refinement: The method currently implemented in Lagrange-Geometry subclass

- - Evaluates integral over element, approximating the integrand by an interpolatory polymomials of two hierarchical orders (2 and 4 at the moment) [**IMAGE HERE**]

- - The running integration error is approximated by the difference between the analytical integrals calculated from these two interpolatory polynomials.

- - The higher order element is split into into sub-elements of lower order, and the integration proceeds recursively.

- - Every time an element is split, its previous running error is subtracted from the total error, and the running errors of the sub-elements are added to the total error. Thus, the integration is terminated when total approximated error is below selected tolerance.

  - using heap structure ordered by the approximate error of the element. This way avoids recursion, and at every iteration selects the element which has worst error, then refines it.

  - When splitting, the previously calculated points are not re-calculated but hierarchically re-used by sub-elements. The sub-element only needs to be refined to a higher hierarchical order, by adding more points.

- *Possible improvement - Performance.* As the the refined element does not check if the neighboring elements are also being refined, so they both sample on the boundary twice. Does there exist a method to store/find intersection refinements faster than just compute 2nd time. Using order 4 for every new refined triangle we sample 9 new points, out of which 2 are being wasted, thus 22% inefficient.

Monte-Carlo integration - according to above mentioned paper, good for dimensions 7 and above.

- Randomly samples function over element, integral is approximated by the average over the sample

- natural error estimate using sample standard deviation

- Stratified Sampling: if after a set number of iterations sample error is larger than expected, then function is highly non-uniform. Split element in equal parts and continue recursively.

- Markov Chain Monte Carlo (MCMC): Uses random walk to sample the integrand, thus concentrating the sample points where the function varies most. Can use Metropolis-Hastings to also adapt the sampling distribution.

Interpolatory Spline integration - this is just another idea...

- Makes grid over element, cubic-interpolates all consecutive partially-overlapping segments, integrates analytically over each segment.

  -tricky part 1: when interpolatory segments intersect, with what weights to take the intersecting parts

-tricky part 2: how well does this method interpolate boundaries of the element (being close to edges and faces)

-tricky part 3: how to estimate error of integration and necessary grid step?

- Any way to do the refinement or error-control?