Reliable Adaptive Algorithms for Integration, Interpolation and Optimization

Fred J. Hickernell
Department of Applied Mathematics, Illinois Institute of Technology
hickernell@iit.edu  mypages.iit.edu/~hickernell

Joint work with
Sou-Cheng Choi (NORC & IIT)
Yuhan Ding (IIT)
Lan Jiang (IIT)
Lluis Antoni Jiménez Rugama (IIT)
Art Owen (Stanford)
Xin Tong (UIC)
Yizhi Zhang (IIT)
Xuan Zhou (J. P. Morgan)

Supported by NSF-DMS-1115392, 1522687

Thank you for your kind invitation to visit!
Why We Want Adaptive Algorithms

% An easy function to integrate ... numerically
f1 = @(x) sin(x.^2);
tic, I1 = integral(f1,0,1), toc
I1 =
    0.310268301723381
Elapsed time is 0.001153 seconds.

% A hard function to integrate ... numerically
f2 = @(x) 200*sin((200*x).^2);
tic, I2 = integral(f2,0,1), toc
I2 =
    0.625850570394814
Elapsed time is 0.016246 seconds.

Adaptive algorithms exert enough computational effort to get the correct answer, but not much more effort than necessary.
Why Adaptive Algorithms Fail

approximating $f_{\text{spiky}}$ with Chebfun
minimizing $f_{\text{spiky}}$ with $\text{fminbnd}$

\[
\int_{0}^{1} f_{\text{fluky}}(x) \, dx = 0.278827
\]

\[
\text{integral}(f_{\text{fluky}}, 0, 1) = 0.278799
\]

meanMC_CLT($Y, 0.01, \ldots$) aims to estimate $\mathbb{E}(Y)$ with an absolute error of 0.01 with 99% confidence. The answer is wrong half the time.
Why Adaptive Algorithms Fail

approximating $f_{spiky}$ with Chebfun
minimizing $f_{spiky}$ with fminbnd

Chebfun (Hale et al., 2015) and fminbnd stop sampling when they think that they have enough information, but miss the spike
Why Adaptive Algorithms Fail

\[ \int_0^1 f_{\text{fluky}}(x) \, dx = 0.278827 \]

\[ \text{integral}(f_{\text{fluky}}, 0, 1) = 0.278799 \]

integral approximates its error by the difference of two different quadratures, which might coincidentally coincide
meanMC_CLT(Y, 0.01, ...) aims to estimate $\mathbb{E}(Y)$ with an absolute error of 0.01 with 99% confidence. The answer is wrong half the time.

Insufficient (100) samples are taken to estimate $\text{var}(Y)$ because $Y$ has a heavy-tailed distribution (large kurtosis).
Why Adaptive Algorithms Fail

Adaptive algorithms fail due to one or both of the following:

- We do not use enough samples to start.
- Our error bounds are faulty.
Why Adaptive Algorithms Fail

Adaptive algorithms fail due to one or both of the following:

- We do not use enough samples to start. How many samples is enough?
- Our error bounds are faulty. What are the alternatives?
Making the Trapezoidal Rule Adaptive

The trapezoidal rule for approximating $\int_0^1 f(x) \, dx$ is

$$T_n(f) = \frac{1}{2n} [f(0) + 2f(1/n) + \cdots + 2f(1 - 1/n) + f(1)].$$

The quadrature error is

$$\left| \int_0^1 f(x) \, dx - T_n(f) \right| \leq \frac{\text{Var}(f')}{{8n^2}}.$$

Without knowing $\text{Var}(f')$, what is $n$ that makes this error $\leq \varepsilon_a$?
Making the Trapezoidal Rule Adaptive

The quadrature error is

\[ \left| \int_0^1 f(x) \, dx - T_n(f) \right| \leq \frac{\text{Var}(f')}{8n^2} \leq \epsilon_a, \]

(ERR)

for \( n = ? \) without knowing \( \text{Var}(f') \).

We assume that \( f \) is not too spiky, i.e.,

\[ \text{Var}(f') \leq \tau \| f' - f(1) + f(0) \|_1, \]

(CONE)

and note that

\[ \| f' - f(1) + f(0) \|_1 \leq \sum_{i=1}^{n} \left| f(i/n) - f((i - 1)/n) - \frac{f(1) - f(0)}{n} \right| + \frac{\text{Var}(f')}{2n} \]

This implies that (ERR) is satisfied for \( f \) satisfying (CONE), and \( n \) satisfying

\[ \frac{\tau}{4n(2n - \tau)} \sum_{i=1}^{n} \left| f(i/n) - f((i - 1)/n) - \frac{f(1) - f(0)}{n} \right| \leq \epsilon_a \]
**integral\_g in GAIL**

This rigorously justified, data-based error bound is employed in the adaptive algorithm \textit{integral\_g}, part of the Guaranteed Automatic Integration Library (GAIL) (Choi et al., 2013–2015).

Clancy et al. (2014) contains details and shows that the computational cost of \textit{integral\_g} is

\[ \Theta \left( \sqrt{\tau \text{Var}(f')} / \varepsilon_a \right) \]

operations. This cost is asymptotically optimal among all possible algorithms. H. et al. (2015+) remove the factor of \( \tau \).

\[ \int_0^1 f_{\text{fluky}}(x) \, dx = 0.278827 \]

\( \text{integral}(f_{\text{fluky}},0,1) = 0.278799 \)

\( \text{integral\_g}(f_{\text{fluky}},0,1) = 0.278827 \)

Don’t approximate error by the difference of two quadratures (Lyness, 1983).
## General Recipe for Justified Adaptive Algorithms

| Error bound | \[ \left| \int_0^1 f(x) \, dx - T_n(f) \right| \leq \frac{\text{Var}(f')}{{8n}^2} \] |
| Cone condition | \[ \text{Var}(f') \leq \tau \|f' - f(1) + f(0)\|_1 \] |
| For \( f \approx A(f) \) | \[ \|f' - f(1) + f(0)\|_1 \leq \|A(f)' - f(1) + f(0)\|_1 + \frac{\text{Var}(f')}{{2n}} \] |
| Data-based error bound | \[ \frac{\tau \|A(f)' - f(1) + f(0)\|_1}{{4n}(2n - \tau)} \] |
| Function Approximation, Optimization, etc. | \[ \|\text{sol}(f) - \text{appx}_n(f)\| \leq \frac{c_1\|f\|}{{np}} \] |
| convex \( \text{sol} \) and \( \text{appx}_n \) | \[ \|f\| \leq \tau|f| \] |
| | \[ |f| \leq |A(f)| + \frac{c_2\|f\|}{{np}} \] |
| so \( f \) | \[ |f| \leq \tau|A(f)| + \frac{c_2\tau\|f\|}{{np}} \] |
| so \( f \) | \[ |f| \leq \frac{\tau|A(f)|}{{1 - c_2\tau/{np}}} \] |
funappx_g and funmin_g in GAIL

GAIL includes rigorously justified, adaptive algorithms for function approximation and function minimization (Choi et al., 2013–2015).

The error bounds employed ensure that enough samples are taken. Spiky functions are handled well provided that \( \tau \) is large enough. See Clancy et al. (2014) and Tong (2014) for details.
Making Monte Carlo Adaptive

Monte Carlo algorithms approximate $\mu = \mathbb{E}(Y) = \int_{-\infty}^{\infty} y \varphi(y) \, dy$ by a sample average:

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} Y_i, \quad Y_1, Y_2, \ldots \overset{\text{iid}}{\sim} Y$$

when the $Y_i$ can be generated, but the integral is too hard. The Central Limit Theorem (CLT) says that

$$\mathbb{P} \left[ \left| \mu - \hat{\mu}_n \right| \leq \frac{2.58 \hat{\sigma}}{\sqrt{n}} \right] \approx 99\%,$$

but this is not rigorous.
Making Monte Carlo Adaptive

\[ \mu = \mathbb{E}(Y) \approx \hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} Y_i, \quad Y_1, Y_2, \ldots \sim \text{iid} Y \]

To obtain a guaranteed error bound we need to assume a bound on the kurtosis of \( Y \):

\[ \text{kurt}(Y) := \mathbb{E}[(Y - \mu)^4]/\text{var}(Y)^2 \leq \kappa_{\text{max}}. \quad \text{(CONE)} \]

Cantelli’s inequality implies a bound on the true variance in terms of the sample variance:

\[ \mathbb{P}[\mathcal{C}^2 \hat{\sigma}^2 \geq \text{var}(Y)] \geq 99.5\%, \quad \text{where} \quad \hat{\sigma}^2 = \frac{1}{n_{\sigma} - 1} \sum_{i=1}^{n_{\sigma}} (Y_i - \hat{\mu}_{n_{\sigma}})^2, \]

where \( \mathcal{C} > 1 \) and \( n_{\sigma} \) depend on \( \kappa_{\text{max}} \). The Berry-Essseen Inequality (a finite sample version of the CLT) determines the sample size, \( n \), needed for the sample mean:

\[ \Phi \left( -\frac{\sqrt{n} \varepsilon_a}{(\mathcal{C} \hat{\sigma})} \right) + \Delta_n \left( \sqrt{n} \varepsilon_a / (\mathcal{C} \hat{\sigma}), \kappa_{\text{max}} \right) \leq 0.0025. \]

Compute the sample mean, \( \hat{\mu}_n \), of an independent sample of size \( n \).
meanMC_g in GAIL

GAIL includes rigorously justified, algorithm meanMC_g for computing $\mu = \mathbb{E}(Y)$ to the desired error tolerance (Choi et al., 2013–2015; H. et al., 2014).

Heavy-tailed random variables are handled well provided that $\kappa_{\text{max}}$ is large enough.

meanMC_CLT(Y, ...) fails half the time
meanMC_g(Y, ...) succeeds 99% of the time

One may use meanMC_g to evaluate

$$\int_{\mathbb{R}^d} g(x) \, dx = \int_{\mathbb{R}^d} f(x) \, \varrho(x) \, dx = \mathbb{E}[f(X)],$$

where $X \sim \varrho$ for some probability density function $\varrho$. 
Making Quasi-Monte Carlo Adaptive

Quasi-Monte Carlo (QMC) algorithms approximate \( \mu = \int_{[0,1]^d} f(x) \, dx \) by a sample average of correlated, evenly placed points. The evenness improves the convergence to the answer (Dick et al., 2014).
Making Quasi-Monte Carlo Adaptive

Quasi-Monte Carlo (QMC) algorithms approximate \( \mu = \int_{[0,1]^d} f(x) \, dx \) by a sample average of correlated, evenly placed points. The evenness improves the convergence to the answer (Dick et al., 2014).
Making Quasi-Monte Carlo Adaptive

Quasi-Monte Carlo (QMC) algorithms approximate \( \mu = \int_{[0,1]^d} f(x) \, dx \) by a sample average of correlated, evenly placed points. The evenness improves the convergence to the answer (Dick et al., 2014).
Making Quasi-Monte Carlo Adaptive

Quasi-Monte Carlo (QMC) algorithms approximate $\mu = \int_{[0,1]^d} f(x) \, dx$ by a sample average of correlated, evenly placed points. The evenness improves the convergence to the answer (Dick et al., 2014).
Making Quasi-Monte Carlo Adaptive

Quasi-Monte Carlo (QMC) algorithms approximate
\[ \mu = \int_{[0,1]^d} f(x) \, dx \]
by a sample average of correlated, evenly placed points. The evenness improves the convergence to the answer (Dick et al., 2014).

Fred J. Hickernell, hickernell@iit.edu • Reliable Adaptive Algorithms • U Illinois Chicago, 10/5/15
Making Quasi-Monte Carlo Adaptive

Quasi-Monte Carlo (QMC) algorithms approximate \( \mu = \int_{[0,1]^d} f(x) \, dx \) by a sample average of correlated, evenly placed points. The evenness improves the convergence to the answer (Dick et al., 2014).
Making Quasi-Monte Carlo Adaptive

Quasi-Monte Carlo (QMC) algorithms approximate \( \mu = \int_{[0,1]^d} f(x) \, dx \) by a sample average of correlated, evenly placed points. The evenness improves the convergence to the answer (Dick et al., 2014).
Data-driven error bounds for quasi-Monte Carlo are obtained via discrete Fourier transforms of the function values (H. and Jiménez Rugama, 2015+; Jiménez Rugama and H., 2015+):

- Fourier Walsh for Sobol’ sampling, and
- Fourier sine/cosine for lattice sampling.

Substantial speedups can by using QMC sampling rather than IID sampling.

```matlab
% Option price class object
% based on user-defined input
MCopt = optPrice(inp);
% meanMC_g to compute the price
tic
MCPrice = genOptPrice(MCopt)
toc
% Results
MCgenoptPrice = 10.021
Elapsed time is 42.688 seconds.

% Now use Sobol' sampling ...
% with a PCA construction
% After changing the ...
% parameters in inp
QMCopt = optPrice(inp)
tic
QMCPrice = genOptPrice(QMCopt)
toc
% Results
QMCgenoptPrice = 10.021
Elapsed time is 0.080 seconds.
```
Summary

- Adaptive algorithms are valuable because they determine the computational effort required based on the problem difficulty.
- Most adaptive algorithms are flawed.
- By moving from error analysis based on properties of input (functions, random variables, etc.) to test, we can justify adaptive algorithms.
- Relative errors can be handled.
- More problems are waiting to be tackled: algorithms with higher order convergence, multi-level Monte Carlo, ... 
- Applied and computational mathematicians should learn to write code well, not only prove theorems about their algorithms.
Raising the Bar for Numerical Algorithms

Implemented

- Widely available in a common language
- Has a convenient user interface
- Tested for bugs
- Coded efficiently
- Stopping criterion is data-based
- Provides an answer within a user-specified error tolerance
- Runs in a reasonable amount of time on available hardware
- All inputs and parameters can be specified explicitly
- Can be written in pseudo-code

Implementable
Raising the Bar for Numerical Algorithms

Justified

- The stopping criterion to reach a user-specified error tolerance is guaranteed to succeed
- This rate of convergence is asymptotically optimal with respect to all possible algorithms
- The rate of convergence as the computational effort tends to infinity is known
- The error can be bounded in terms of information about the inputs and parameters
- Has a theoretical basis
- Provides a reasonable answer for a wide range of problems

Justifiable
Thank you


