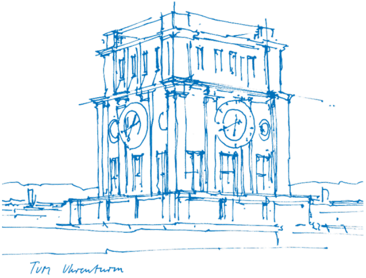
Algorithms for Scientific Computing

Hierarchical Methods and Sparse Grids
– Archimedes' Quadrature, High-Dimensional –

Michael Bader
Technical University of Munich

Summer 2022



Numerical Quadrature (So Far . . .)

- Hierarchical and non-hierarchical one-dimensional quadrature
- Aim: dealing with high-dimensional functions
- Quadrature as an example: well-studied, relatively simple

- On the way to high dimensionalities we have to consider whether effort (measured in function evaluations, computations, . . .) is well-invested?

$\Rightarrow$ Consider ratio of cost vs. accuracy

Part I

Cost and Accuracy

ϵ-Complexity of Numerical Methods

Relate Cost to Achieved Accuracy:

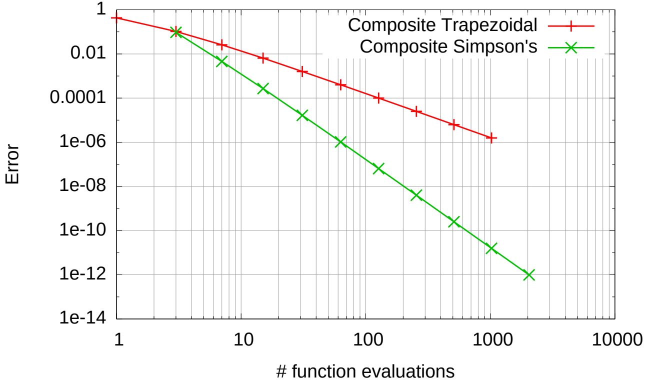
- Usually approximate solution with **error ϵ**
  (due to discretization, rounding, truncation, ...)
- To measure cost W: count operations (function evaluations, e.g.)
- Relate cost W to error ϵ
  $\Rightarrow$ **How many operations $W(\epsilon)$ to obtain error of at most ϵ?**

Example: Composite Integration Rules

- Composite Trapezoidal (CT) rule with n subintervals:
  - n+1 function evaluations
  - Error $\mathcal{O}(n^{-2})$ (sufficiently smooth)
  - ϵ-complexity $W(\epsilon) = \mathcal{O}(\sqrt{1/\epsilon})$ [function evaluations]
- Composite Simpson's (CS) rule correspondingly $W(\epsilon) = \mathcal{O}(\sqrt[4]{1/\epsilon})$

CT and CS: Cost-Error Diagram

- $F_1 := \int_0^\pi \sin(x)\,dx$, determine $|CT - F_1|$ and $|CS - F_1|$



- ϵ-complexities $\mathcal{O}(\sqrt{1/\epsilon})$ and $\mathcal{O}(\sqrt[4]{1/\epsilon})$

$\rightsquigarrow$ Different gradients of the curves
(asymptotically for large n; double-logarithmic scale)

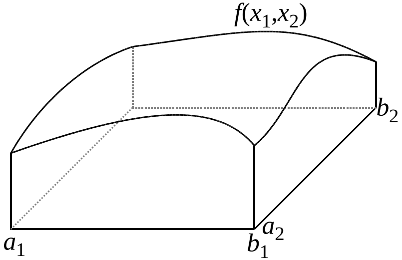
Multi-Dimensional Quadrature

- Now on to multi-dimensional functions:

$$\text{Area of integration } \Omega := \prod_{k=1}^{d} [a_k, b_k], \text{ function } f : \Omega \to \mathbb{R}$$

- Compute approximation for

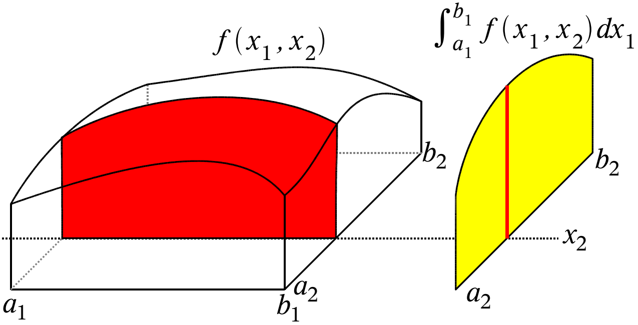
$$F_d(f, \Omega) := \int_{\Omega} f(x_1, \ldots, x_d)\, d\vec{x}.$$



Decomposition into One-Dimensional Integrals

- Decompose d-dimensional integral into sequence of one-dimensional ones (cf. Fubini's Theorem)

$$F_d(f,\Omega) = \int_{a_d}^{b_d} \cdots \int_{a_2}^{b_2} \left(\int_{a_1}^{b_1} f(x_1,\ldots,x_d)\, dx_1 \right) dx_2 \ldots dx_d.$$



Decomposition: Implementation

- Consider this decomposition using the function F_1 (one-dimensional integration), and functions G_k:

$$
\begin{aligned}
G_0(x_1, x_2, x_3, \ldots, x_d) &:= f(x_1, x_2, x_3, \ldots, x_d) \\
G_1(x_2, x_3, \ldots, x_d) &:= F_1(G_0(\bullet, x_2, x_3, \ldots, x_d), a_1, b_1) \\
G_2(x_3, \ldots, x_d) &:= F_1(G_1(\bullet, x_3, \ldots, x_d), a_2, b_2) \\
&\vdots \\
G_d() &:= F_1(G_{d-1}(\bullet), a_d, b_d)
\end{aligned}
$$

- G_k integrates over $x_1, \ldots, x_k$; remaining variables free

Numerical quadrature

- Replace F_1 by a quadrature formula, such as CT, CS, ...

Cost and Accuracy

Cost

- Uniform grid with n subintervals for 1d quadrature
- d dimensions: Cartesian product of 1d grids
- Indices

$$(i_1, \ldots, i_d) \in \{0, 1, 2, \ldots, n\}^d$$

  with corresponding grid points

$$(x_1, \ldots, x_d) \text{ with } x_k = a_k + i_k \frac{b_k - a_k}{n}$$

- Total cost (composite trapezoidal sum):
  - $(n+1)^d$ (with grid points on domain's boundary $\partial\Omega$)
  - $(n-1)^d$ (if f is zero on $\partial\Omega$)

Cost and Accuracy (2)

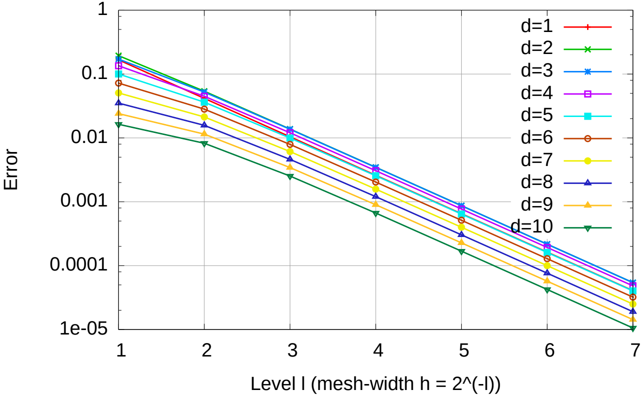
Accuracy

- Still $\mathcal{O}(n^{-2})$ for CT, $\mathcal{O}(n^{-4})$ for CS
- Remark: starting with G_2, the current function values are erroneous by $\mathcal{O}(n^{-2})$ and $\mathcal{O}(n^{-4})$ resp.; this does not alter the overall accuracy

$\Rightarrow$ Thus everything is fine...?

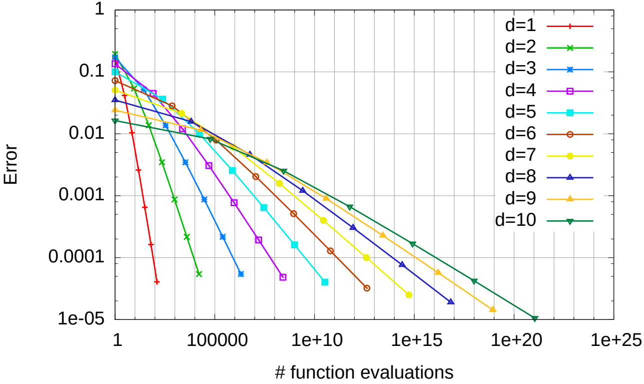
Multidimensional Quadrature: Example

- Integration of
$$f(x_1, \ldots, x_d) := \prod_{k=1}^{d} 4x_k(1 - x_k)$$
on $\Omega = [0, 1]^d$ with the composite Trapezoidal rule
- Error:



Multidimensional Quadrature: Example (2)

- For ϵ-complexity:
  Use cost (number of function evaluations) as abscissa



- Does not look that good any more…

Multidimensional Quadrature: Example (3)

"10^{21} function evaluations":

- Large number…
- 1 ZFlop (Zeta) = 1.000.000.000.000 GFlop = 1.000.000 PFlop (if only one op. per grid point)
- Compute on LRZ's supercomputer SuperMUC-NG:
  - Peak performance: $\approx$ 25 PFlop/s
- It would take approx. 1/2 day to compute the integral, assuming that one function evaluation takes only one floating-point operation, and one one floating-point operation can be performed in each clock cycle…

Curse of Dimensionality

ϵ-complexity

- CT: $\mathcal{O}(\epsilon^{-\frac{d}{2}})$, CS: $\mathcal{O}(\epsilon^{-\frac{d}{4}})$

Curse of dimensionality

- Exponential dependency on dimensionality d
- Higher-dimensional problems infeasible to tackle ($d = 10$ is still moderate ...)
- Property of the problem – or just of the algorithm?
- It's the algorithm $\Rightarrow$ hierarchical methods (among few others) will be able to mitigate the curse of dimensionality to some extent

Monte-Carlo Integration

- example for a better methods for numerical quadrature:
- simple approach, simple to implement

Monte-Carlo Idea:

- X be a random variable, uniformly distributed on Ω
- The expectation of X is then given as

$$E(f(X)) = \frac{1}{\mathrm{Vol}(\Omega)} \int_\Omega f(x)\, dx = \frac{1}{\mathrm{Vol}(\Omega)} F_d(f, \Omega)$$

- On the other hand: if x_k are realizations of X we obtain

$$\lim_{M\to\infty} \frac{1}{M} \sum_{k=1}^{M} f(x_k) = E(f(X))$$

with probability 1 (strong law of large numbers)

Monte-Carlo Integration (2)

- Simple to implement
- Cost completely independent of d (counting function evaluations)
- Accuracy?
  - Estimate stochastically: compute standard deviation (use additivity of variances)

$$\sqrt{\mathrm{Var}\left(\frac{1}{M}\sum_{k=1}^{M} f(x_k)\right)} = \sqrt{\frac{1}{M^2}\sum_{k=1}^{M}\mathrm{Var}(f)} = \sqrt{\frac{\mathrm{Var}(f)}{M}}$$

  - Independent of d, too
  - Dependencies of d only in $\mathrm{Var}(f)$ and $\mathrm{Vol}(\Omega)$ possible; does not affect exponent of M
- Thus (stochastically) ϵ-complexity of $\mathcal{O}(\epsilon^{-2})$
  - Very slow convergence, but independent of d
  - thus: very helpful for tackling high-dimensional problems!

What Next?

- We know that the curse of dimensionality can be overcome
- Search for alternative (better?) methods
  $\rightsquigarrow$ which can be used for other applications apart from integration as well, for example
- approach: hierarchical bases in higher dimensions

Part II

Archimedes, *d*-Dimensional

Current State: One-Dimensional Quadrature

- One-dimensional functions f, interval $[a, b]$
- Compute approximation $F_1(f, a, b)$ of area:

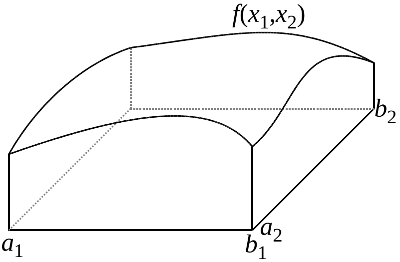
$$F_1(f, a, b) \approx \int_a^b f(x)\, dx$$

- Notation for appoximation of exact integral value in the following: $F_d(.)$, with d as the dimension
- One-dimensional quadrature rules:
  - Composite trapeziodal rule
  - Composite Simpson's rule
  - Archimedes' quadrature

Multi-Dimensional Quadrature

Consider multi-dimensional setting

$$F_d(f,\Omega) \approx \int_\Omega f(x_1,\ldots,x_d)\,d\vec{x}, \qquad \Omega := \prod_{k=1}^{d}[a_k,b_k]$$



First Attempt

- remember theorem of Fubini:

$$F_d(f,\Omega) = \int_{a_d}^{b_d} \cdots \int_{a_1}^{b_1} f(x_1,\ldots,x_d)\, dx_1 \ldots dx_d$$

- Use full-grid approach as before:

$$\begin{aligned}
G_0(x_1, x_2, x_3, \ldots, x_d) &:= f(x_1, x_2, x_3, \ldots, x_d) \\
G_1(x_2, x_3, \ldots, x_d) &:= F_1(G_0(\bullet, x_2, x_3, \ldots, x_d), a_1, b_1) \\
G_2(x_3, \ldots, x_d) &:= F_1(G_1(\bullet, x_3, \ldots, x_d), a_2, b_2) \\
&\vdots \\
G_d() &:= F_1(G_{d-1}(\bullet), a_d, b_d)
\end{aligned}$$

- We now consider the effect of Archimedes' quadrature as one-dimensional quadrature method for F_1

First Attempt: Employing Archimedes

- d nested loops $(x_1, x_2, \dots)$
- Summation of weighted function values
- No real advantages apart from adaptivity
  (which is not very useful this way)

Interplay of hierarchization and summation (integration)

- Consider setting with $d = 2$
- First, compute integrals in x_1-direction: $F_1(G_0(\bullet, x_2), a_1, b_1)$
  - Involves hierarchization in x_1-direction
  - But no impact on $G_1(x_2)$
- $G_1(x_2)$: no hierarchical values, thus all $G_1(x_2)$ of same order
- After summation (integration) in x_1-direction:
  - Hierarchization in x_2-direction
  - Finally summation in x_2-direction

Improved Version

- Consider computing $G_1(x_2)$
  - We are only interested in hierarchical surplus
  - Hierarchical surplus typically much smaller than function value

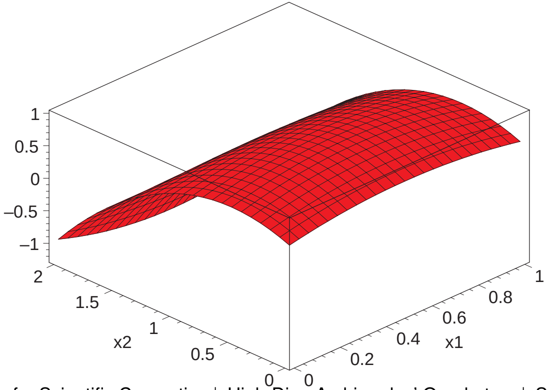
  $\Rightarrow$ Could be computed with much less grid points in x_1-direction
- We change the order of “integration in x_1-direction” and “hierarchization in x_2-direction”
  - Write hierarchical area elements of quadrature in x_2-direction (trapezoid, segments, triangles) as function of x_1
  - Integrate those in x_1-direction
- Now interplay of dimensions for integration much more complicated
- … but this will lead to much more efficient method

Example, $2d$

Consider

$$f(x_1, x_2) := \left(x_1 + \frac{1}{2}\right)\left(x_1 - \frac{3}{2}\right)\left(x_2 + \frac{1}{2}\right)\left(x_2 - \frac{3}{2}\right)$$

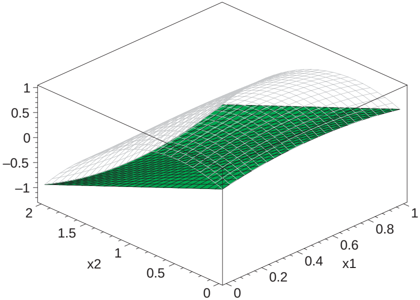
on $\Omega = [0, 1] \times [0, 2]$

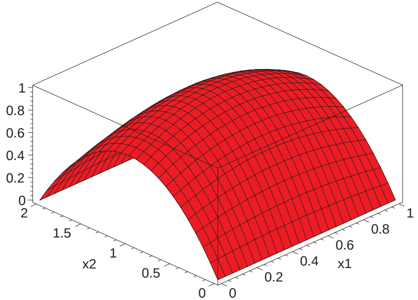


Trapezoidal Volume and Remainder Segment

First step of the hierarchical decomposition

$$F_2(f,\Omega) = F_1(T_2, a_1, b_1) + S_2(f,\Omega)$$





“Green function” $\rightarrow$ linear interpolation of values at a_2, b_2:

$$\frac{f(x_1,a_2)(b_2-x_2)+f(x_1,b_2)(x_2-a_2)}{b_2-a_2} \qquad \text{for any } x_1$$

Trapezoidal Volume and Remainder Segment (2)

Decompose volume into

- trapezoidal (for constant x_1) cross-section with area

$$T_2(x_1) := \frac{b_2 - a_2}{2}\left(f(x_1, a_2) + f(x_1, b_2)\right),$$

$\rightarrow$ to be integrated in x_1-direction using quadrature rule F_1

- and remainder segment

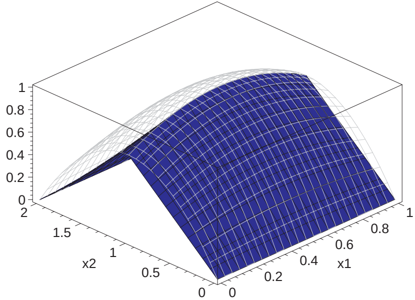
$$S_2(f, \Omega) := F_2(f, \Omega) - F_1(T_2, a_1, b_1)$$
$$= \int_{a_2}^{b_2}\int_{a_1}^{b_1} \left(f(x_1, x_2) - \frac{f(x_1, a_2)(b_2 - x_2) + f(x_1, b_2)(x_2 - a_2)}{b_2 - a_2} \right) dx_1\, dx_2$$

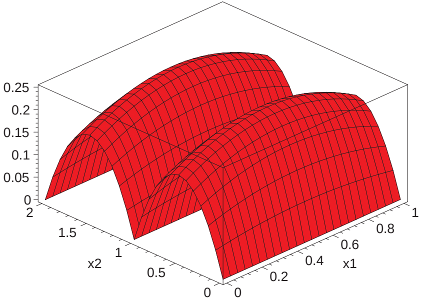
Note: T_2 is the integral over the linear interpolation ("green function")

Triangular Volumes and Remainder Segments

Second step of the hierarchical decomposition

$$S_2(f,\Omega) = F_1(D_2, a_1, b_1) + S_2(f,\ldots) + S_2(f,\ldots)$$





again: hierarchization in x_2-direction; integrate in x_1-direction

Triangular Volumes and Remainder Segments (2)

Decompose remainder segment $S_2(f, \Omega)$ into

- triangular (for constant x_1) cross-section with area

$$D_2(x_1) := \frac{b_2 - a_2}{2} \left(f\left(x_1, \frac{a_2 + b_2}{2}\right) - \frac{f(x_1, a_2) + f(x_1, b_2)}{2} \right)$$

$\rightarrow$ to be integrated in x_1-direction using quadrature rule F_1

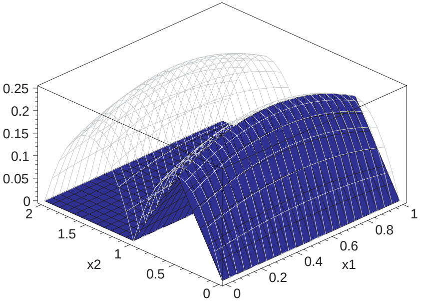
- and two remainder segments

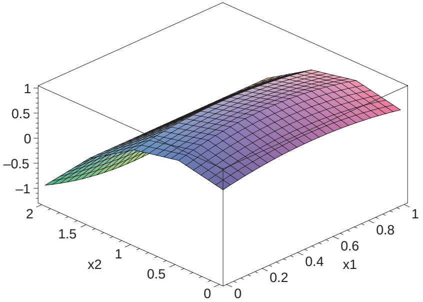
$$\begin{aligned} S_2(f, [a_1, b_1] \times [a_2, b_2]) &= F_1(D_2, a_1, b_1) \\ &+ S_2\left(f, [a_1, b_1] \times \left[a_2, \frac{a_2 + b_2}{2}\right]\right) \\ &+ S_2\left(f, [a_1, b_1] \times \left[\frac{a_2 + b_2}{2}, b_2\right]\right) \end{aligned}$$

Triangular Volumes and Remainder Segments (3)

Recursive decomposition

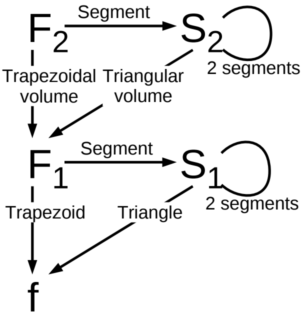
- Repeat last step for both remainder segments
- Decompose each into triangular sub-volume and two remainder segments
- Example for one of the two segments and sum of trapezoidal and first three triangular sub-volumes:





Recursive Structure of Function Calls

- Nested recursive structure of function calls
- For higher-dimensional problems: one more level (F_d and S_d) for each additional dimension



- Consider number of function evaluations for grid point inside of Ω
  - Straightforward: 3^d evaluations to compute surplus
  - All but one have already been computed!

Subvolumes

- F_1: the subvolumes (hierarchized in x_2-direction) are decomposed (in x_1-direction) into trapezoid and many triangles
- Integrand itself is area (one slice trapezoidal/triangular subareas)
- Subvolumes which are added in quadrature are pagodas (neglecting trapezoidals)
  - Height of pagodas: d-dimensional hierarchical surplus
  - Volume of pagodas: 2^{-d} times size of support times surplus (more in next part)
- Taking stopping criterion depending on surplus (d criteria: one for each S_i)
  - Find those grid points for which function evaluation is worthwile
  - In general much less than naive implementation
- Extend from composite trapezoidal rule to Simpsons' as in one-dimensional setting

Archimedes Quadrature – d Dimensions

→ Summary of the Algorithm

Start of recursion → “trapezoid plus segment S”:

$$\begin{aligned}
&F_d^{\text{Arch}}\big(f(x_1,\dots,x_d),[a_1,b_1]\times\dots\times[a_d,b_d]\big)\\
&= F_{d-1}^{\text{Arch}}\left(T_d(x_1,\dots,x_{d-1}),[a_1,b_1]\times\dots\times[a_{d-1},b_{d-1}]\right)\\
&+ S_d\big(f(x_1,\dots,x_d),[a_1,b_1]\times\dots\times[a_d,b_d]\big)
\end{aligned}$$

with “trapezoid” function

$$T_d(x_1,\dots,x_{d-1}) = \tfrac{b_d-a_d}{2}\big(f(x_1,\dots,x_{d-1},a_d)+f(x_1,\dots,x_{d-1},b_d)\big)$$

Archimedes Quadrature – d Dimensions

→ **Summary of the Algorithm (2)**

Dimensional recursion for surplus section S:

$$
\begin{aligned}
& S_d\big(f(x_1,\ldots,x_d),[a_1,b_1]\times[a_2,b_2]\times\cdots\times[a_d,b_d]\big)\\
&= F^{\text{Arch}}_{d-1}\big(D_d(x_1,\ldots,x_{d-1}),[a_1,b_1]\times\cdots\times[a_{d-1},b_{d-1}]\big)\\
&+ S_d\left(f(x_1,\ldots,x_d),[a_1,b_1]\times\cdots\times[a_{d-1},b_{d-1}]\times\left[a_d,\tfrac{a_d+b_d}{2}\right]\right)\\
&+ S_d\left(f(x_1,\ldots,x_d),[a_1,b_1]\times\cdots\times[a_{d-1},b_{d-1}]\times\left[\tfrac{a_d+b_d}{2},b_d\right]\right)
\end{aligned}
$$

with $D_d(x_1,\ldots,x_{d-1}) =$

$$\frac{b_d-a_d}{2}\left(f\left(x_1,\ldots,x_{d-1},\frac{a_d+b_d}{2}\right)-\frac{f(x_1,\ldots,x_{d-1},a_d)+f(x_1,\ldots,x_{d-1},b_d)}{2}\right)$$