

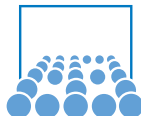
# Algorithms of Scientific Computing

## Hierarchical Methods and Sparse Grids

Michael Bader

Technische Universität München

Summer Term 2013



# Part III

## Cost and Accuracy

## 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?
- ⇒ Consider ratio of cost vs. accuracy

# $\epsilon$ -Complexity

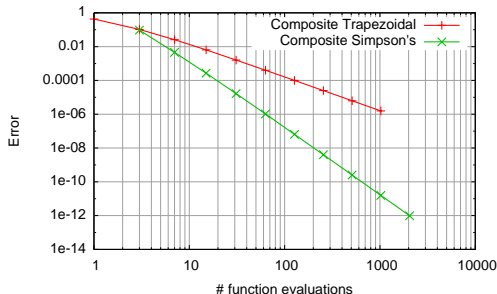
- Numerical methods: usually approximate solution with error  $\epsilon$ 
  - Error can be due to discretization, rounding, truncation, . . .
- To measure cost  $W$ : count operations
- Relate cost  $W$  to error  $\epsilon$ 
  - How many operations  $W(\epsilon)$  required to obtain error of at most  $\epsilon$ ?
- To this end: assumptions about solution again (differentiability, bounds for derivatives, . . . )
  - Often don't hold in real-world settings
  - But good indication to compare different methods

# $\epsilon$ -Complexity

- Numerical methods: usually approximate solution with error  $\epsilon$ 
  - Error can be due to discretization, rounding, truncation, ...
- To measure cost  $W$ : count operations
- Relate cost  $W$  to error  $\epsilon$ 
  - How many operations  $W(\epsilon)$  required to obtain error of at most  $\epsilon$ ?
- To this end: assumptions about solution again (differentiability, bounds for derivatives, ...)
  - Often don't hold in real-world settings
  - But good indication to compare different methods
- Composite trapezoidal rule with  $n$  subintervals:
  - $n+1$  evaluations
  - Error  $\mathcal{O}(n^{-2})$  (sufficiently smooth)
  - $\epsilon$ -complexity  $W(\epsilon) = \mathcal{O}(\sqrt{1/\epsilon})$  (unit: number of evaluations)
- Composite Simpson's rule correspondingly  $W(\epsilon) = \mathcal{O}(\sqrt[4]{1/\epsilon})$

# CT and CS: Example

- Cost-error diagram for  $F_1 := \int_0^\pi \sin(x) dx$ :  
 $|CT - F_1|$  and  $|CS - F_1|$
- No function evaluations on the boundary



- $\epsilon$ -complexities  $\mathcal{O}(\sqrt{1/\epsilon})$  and  $\mathcal{O}(\sqrt[4]{1/\epsilon})$
- ↪ Different gradients of the curves (asymptotically for large  $n$ ;  
double-logarithmic scale)

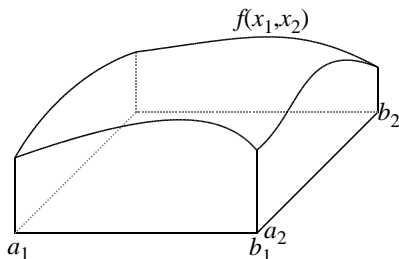
# Multi-Dimensional Quadrature

- Now on to multi-dimensional functions:

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

- Compute approximation for

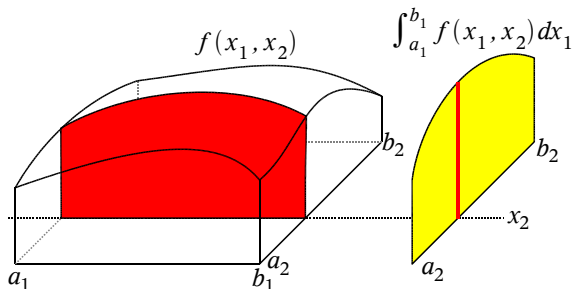
$$F_d(f, \Omega) := \int_{\Omega} f(x_1, \dots, x_d) d\vec{x}.$$



# Decomposition into One-Dimensional Integrals

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

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





# Decomposition: Implementation

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

$$G_0(x_1, x_2, x_3, \dots, x_d) := f(x_1, x_2, x_3, \dots, x_d)$$

# Decomposition: Implementation

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

$$\begin{aligned}G_0(x_1, x_2, x_3, \dots, x_d) &:= f(x_1, x_2, x_3, \dots, x_d) \\G_1(x_2, x_3, \dots, x_d) &:= F_1(G_0(\bullet, x_2, x_3, \dots, x_d), a_1, b_1)\end{aligned}$$

# Decomposition: Implementation

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

$$\begin{aligned}G_0(x_1, x_2, x_3, \dots, x_d) &:= f(x_1, x_2, x_3, \dots, x_d) \\G_1(x_2, x_3, \dots, x_d) &:= F_1(G_0(\bullet, x_2, x_3, \dots, x_d), a_1, b_1) \\G_2(x_3, \dots, x_d) &:= F_1(G_1(\bullet, x_3, \dots, x_d), a_2, b_2)\end{aligned}$$

# Decomposition: Implementation

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

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

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

# Decomposition: Implementation

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

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

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

## Numerical quadrature

- Just replace  $F_1$  by a quadrature formula, e.g. CT, CS

# Cost and Accuracy

## Cost

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

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

with corresponding grid points

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

- Total cost:
  - $(n + 1)^d$  (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

⇒ Thus everything is fine...?

# Multidimensional Quadrature: Example

- Integration of

$$f(x_1, \dots, x_d) := \prod_{k=1}^d 4x_k(1 - x_k)$$

on  $\Omega = [0, 1]^d$  with the composite Trapezoidal rule



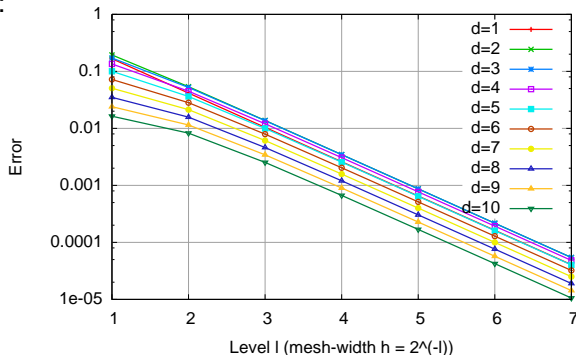
# Multidimensional Quadrature: Example

- Integration of

$$f(x_1, \dots, x_d) := \prod_{k=1}^d 4x_k(1 - x_k)$$

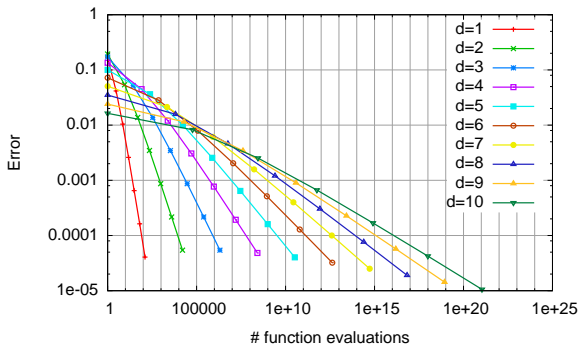
on  $\Omega = [0, 1]^d$  with the composite Trapezoidal rule

- Error:



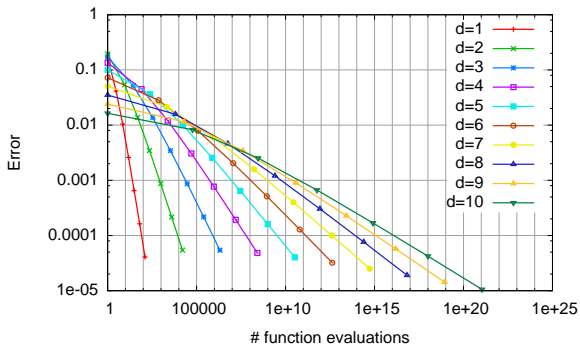
## Multidimensional Quadrature: Example (2)

- Having  $\epsilon$ -complexity in mind:
  - Use cost (number of function evaluations) as abscissa



## Multidimensional Quadrature: Example (2)

- Having  $\epsilon$ -complexity in mind:
  - Use cost (number of function evaluations) as abscissa



- Does not look that good any more. . .

## Multidimensional Quadrature: Example (3)

$10^{21}$

- Large number...

## Multidimensional Quadrature: Example (3)

$10^{21}$

- Large number. . .
- 1 ZFlop (Zeta) = 1.000.000.000.000 GFlop = 1.000.000 PFlop (if only one op. per grid point)

## Multidimensional Quadrature: Example (3)

$10^{21}$

- Large number. . .
- 1 ZFlop (Zeta) = 1.000.000.000.000 GFlop = 1.000.000 PFlop (if only one op. per grid point)
- Compare LRZ's new super computer SuperMUC:
  - Peak performance: 3 PFlop/s
- It would take almost 4 days to compute the integral, assuming that one integration operation can be performed in one clock cycle. . .

# Curse of Dimensionality

## $\epsilon$ -complexity

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

# Curse of Dimensionality

## $\epsilon$ -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) can 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 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  $\Omega$
- The expectation of  $X$  is then given as

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

# 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  $\Omega$
- The expectation of  $X$  is then given as

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

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

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

with probability 1 (strong law of large numbers)

## Monte-Carlo Integration (2)

- Simple to implement
- Cost completely independent of  $d$  (counting function evaluations)

## 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)

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

- Independent of  $d$ , too
- Dependencies of  $d$  only in  $\text{Var}(f)$  and  $\text{Vol}(\Omega)$  possible; does not affect exponent of  $M$

## 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)

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

- Independent of  $d$ , too
- Dependencies of  $d$  only in  $\text{Var}(f)$  and  $\text{Vol}(\Omega)$  possible; does not affect exponent of  $M$
- Thus (stochastically)  $\epsilon$ -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
  - ... which can be used for other applications apart from integration as well, for example
- approach: hierarchical bases in higher dimensions