

MODERN COMPUTATIONAL FINANCE

AAD AND PARALLEL SIMULATIONS

with professional implementation in C++

ANTOINE SAVINE

Preface by Leif Andersen

WILEY

Modern Computational Finance

Modern Computational Finance

AAD and Parallel Simulations

with professional implementation in C++

ANTOINE SAVINE

Preface by Leif Andersen

WILEY

Copyright © 2019 by John Wiley & Sons, Inc. All rights reserved.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600, or on the Web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at www.wiley.com/go/permissions.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993, or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Cataloging-in-Publication Data

Names: Savine, Antoine, 1970- author.

Title: Modern computational finance : AAD and parallel simulations / Antoine Savine.

Description: Hoboken, New Jersey : John Wiley & Sons, Inc., [2019] | Includes bibliographical references and index. |

Identifiers: LCCN 2018041510 (print) | LCCN 2018042197 (ebook) | ISBN 9781119539544 (Adobe PDF) | ISBN 9781119539520 (ePub) | ISBN 9781119539452 (hardcover)

Subjects: LCSH: Finance—Mathematical models. | Finance—Computer simulation. | Automatic differentiation.

Classification: LCC HG106 (ebook) | LCC HG106 .S28 2019 (print) | DDC 332.01/5195—dc23

LC record available at <https://lcn.loc.gov/2018041510>

Cover Design: Wiley

Cover Image: ©kentarcajuan/E+/Getty Images

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

To my wife Audrey, who taught me to love.

To my children, Simon, Sarah, and Anna, who taught me to care.

To Bruno Dupire, who taught me to think.

To Jesper Andreasen, who believed in me when nobody else would.

*And to my parents, Svetlana and Andre Savine,
who I wish were with us to read my first book.*

—Antoine Savine

Contents

Modern Computational Finance	xi
Preface by Leif Andersen	xv
Acknowledgments	xix
Introduction	xxi
About the Companion C++ Code	xxv
PART I	
Modern Parallel Programming	1
Introduction	3
CHAPTER 1	
Effective C++	17
CHAPTER 2	
Modern C++	25
2.1 Lambda expressions	25
2.2 Functional programming in C++	28
2.3 Move semantics	34
2.4 Smart pointers	41
CHAPTER 3	
Parallel C++	47
3.1 Multi-threaded Hello World	49
3.2 Thread management	50
3.3 Data sharing	55
3.4 Thread local storage	56
	vii

3.5	False sharing	57
3.6	Race conditions and data races	62
3.7	Locks	64
3.8	Spinlocks	66
3.9	Deadlocks	67
3.10	RAII locks	68
3.11	Lock-free concurrent design	70
3.12	Introduction to concurrent data structures	72
3.13	Condition variables	74
3.14	Advanced synchronization	80
3.15	Lazy initialization	83
3.16	Atomic types	86
3.17	Task management	89
3.18	Thread pools	96
3.19	Using the thread pool	108
3.20	Debugging and optimizing parallel programs	113
PART II		
Parallel Simulation		123
Introduction		125
CHAPTER 4		
Asset Pricing		127
4.1	Financial products	127
4.2	The Arbitrage Pricing Theory	140
4.3	Financial models	151
CHAPTER 5		
Monte-Carlo		185
5.1	The Monte-Carlo algorithm	185
5.2	Simulation of dynamic models	192
5.3	Random numbers	200
5.4	Better random numbers	202
CHAPTER 6		
Serial Implementation		213
6.1	The template simulation algorithm	213
6.2	Random number generators	223
6.3	Concrete products	230
6.4	Concrete models	245

6.5	User interface	263
6.6	Results	268
CHAPTER 7		
	Parallel Implementation	271
7.1	Parallel code and skip ahead	271
7.2	Skip ahead with mrg32k3a	276
7.3	Skip ahead with Sobol	282
7.4	Results	283
PART III		
Constant Time Differentiation		285
Introduction		287
CHAPTER 8		
	Manual Adjoint Differentiation	295
8.1	Introduction to Adjoint Differentiation	295
8.2	Adjoint Differentiation by hand	308
8.3	Applications in machine learning and finance	315
CHAPTER 9		
	Algorithmic Adjoint Differentiation	321
9.1	Calculation graphs	322
9.2	Building and applying DAGs	328
9.3	Adjoint mathematics	340
9.4	Adjoint accumulation and DAG traversal	344
9.5	Working with tapes	349
CHAPTER 10		
	Effective AAD and Memory Management	357
10.1	The Node class	359
10.2	Memory management and the Tape class	362
10.3	The Number class	379
10.4	Basic instrumentation	398
CHAPTER 11		
	Discussion and Limitations	401
11.1	Inputs and outputs	401
11.2	Higher-order derivatives	402

11.3	Control flow	402
11.4	Memory	403
CHAPTER 12		
	Differentiation of the Simulation Library	407
12.1	Active code	407
12.2	Serial code	409
12.3	User interface	417
12.4	Serial results	424
12.5	Parallel code	426
12.6	Parallel results	433
CHAPTER 13		
	Check-Pointing and Calibration	439
13.1	Check-pointing	439
13.2	Explicit calibration	448
13.3	Implicit calibration	475
CHAPTER 14		
	Multiple Differentiation in <i>Almost</i> Constant Time	483
14.1	Multidimensional differentiation	483
14.2	Traditional Multidimensional AAD	484
14.3	Multidimensional adjoints	485
14.4	AAD library support	487
14.5	Instrumentation of simulation algorithms	494
14.6	Results	499
CHAPTER 15		
	Acceleration with Expression Templates	503
15.1	Expression nodes	504
15.2	Expression templates	507
15.3	Expression templated AAD code	524
	Debugging AAD Instrumentation	541
	Conclusion	547
	References	549
	Index	555

Modern Computational Finance

Computational concerns, the ability to calculate values and risks of derivatives portfolios practically and in reasonable time, have always been a major part of quantitative finance. With the rise of bank-wide regulatory simulations like CVA and capital requirements, it became a matter of survival. Modern computational finance makes the difference between calculating CVA risk overnight in large data centers and praying that they complete by morning, or in real-time, within minutes on a workstation.



Computational finance became a key skill, now expected from all quantitative analysts, developers, risk professionals, and anyone involved with financial derivatives. It is increasingly taught in masters programs in finance, such as the Copenhagen University's MSc Mathematics - Economics, where this publication is the curriculum in numerical finance.

The Danske Bank logo, consisting of the word 'Danske' in white on a dark blue background and the word 'Bank' in blue on a white background with a thin blue border.

Danske Bank's quantitative research built its front office and regulatory systems combining technologies such as model hierarchies, scripting of transactions, parallel Monte-Carlo, a special application of regression proxies, and Automatic Adjoint Differentiation (AAD).

In 2015, Danske Bank demonstrated the computation of a sizeable CVA on a laptop in seconds, and its full market risk in minutes, without loss of accuracy, and won the In-House System of the Year Risk award.



Wiley's Computational Finance series, written by some of the very people who wrote Danske Bank's systems, offers a unique insight into the modern implementation of financial models. The volumes combine financial modeling, mathematics, and programming to resolve real-life financial problems and produce effective derivatives software.

The scientific, financial, and programming notions are developed in a pedagogical, self-contained manner. The publications are inseparable from the professional source code in C++ that comes with them. The books build the libraries step by step and the code demonstrates the practical application of the concepts discussed in the publications.

This is an essential reading for developers and analysts, risk managers, and all professionals involved with financial derivatives, as well as students and teachers in Masters and PhD programs in finance.

ALGORITHMIC ADJOINT DIFFERENTIATION

This volume is written by Antoine Savine, who co-wrote Danske Bank's parallel simulation and AAD engines, and teaches volatility and computational finance in Copenhagen University's MSc Mathematics - Economics.

Arguably the strongest addition to numerical finance of the past decade, Algorithmic Adjoint Differentiation (AAD) is the technology implemented in modern financial software to produce thousands of accurate risk sensitivities within seconds on light hardware. AAD is one of the greatest algorithms of the 20th century. It is also notoriously hard to learn.

This book offers a one-stop learning and reference resource for AAD, its practical implementation in C++, and its application in finance. AAD is explained step by step across chapters that gently lead readers from the theoretical foundations to the most delicate areas of an efficient implementation, such as memory management, parallel implementation, and acceleration with expression templates.

The publication comes with a self-contained, complete, general-purpose implementation of AAD in standard modern C++. The AAD library builds on the latest advances in AAD research to achieve remarkable speed. The code is incrementally built throughout the publication, where all the implementation details are explained.

The publication also covers the application of AAD to financial derivatives and the design of generic, parallel simulation libraries. Readers with working knowledge of derivatives and C++ will benefit most, although the book does cover modern and parallel C++.

The book comes with a professional parallel simulation library in C++, connected to AAD. Some of the most delicate applications of AAD to finance, such as the differentiation through calibration, are also explained in words, mathematics, and code.

Preface by Leif Andersen

It is now 2018, and the global quant community is realizing that size does matter: big data, big models, big computing grids, big computations – and a big regulatory rulebook to go along with it all. Not to speak of the *big* headaches that all this has induced across Wall Street.

The era of “big finance” has been creeping up on the banking industry gradually since the late 1990s, and got a boost when the Financial Crisis of 2007–2009 exposed a variety of deep complexities in the workings of financial markets, especially in periods of stress. Not only did this lead to more complicated models and richer market data with an explosion of basis adjustments, it also emphasized the need for more sophisticated governance as well as quoting and risk managements practices. One poster child for these developments is the new market practice of incorporating portfolio-level funding, margin, liquidity, capital, and credit effects (collectively known as “XVAs”) into the prices of even the simplest of options, turning the previously trivial exercise of pricing, say, a plain-vanilla swap into a cross-asset high-dimensional modeling problem that requires PhD-level expertise in computing and model building. Regulators have contributed to the trend as well, with credit capital calculation requirements under Basel 3 rules that are at the same level of complexity as XVA calculations, and with the transparency requirements of MiFID II requiring banks to collect and disclose vast amounts of trade data.

To get a quick sense of the computational effort involved in a basic XVA calculation, consider that such a computation typically involves path-wise Monte Carlo simulation of option trade prices through time, from today’s date to the final maturity of the trades. Let us say that 10,000 simulations are used, running on a monthly grid for 10 years. As a good-sized bank probably has in the neighborhood of 1,000,000 options on its books, calculating a single XVA adjustment on the bank’s derivatives holding will involve in the order of $10^3 \cdot 10 \cdot 12 \cdot 10^6 \approx 10^{11}$ option re-pricings, on top of the often highly significant effort of executing the Monte Carlo simulation of market data required for pricing in the first place. Making matters significantly worse is then the fact that the traders and risk managers looking after the XVA positions will always require that sensitivities (i.e., partial derivatives) with respect to key risk factors in the market data are returned along with the XVA number itself. For complex portfolios, the number of sensitivities that

one needs to compute can easily be in the order of 10^3 ; if these are computed naively (e.g., by finite difference approximations), the number of option re-pricings needed will then grow to a truly unmanageable order of 10^{14} .

There are many interesting ways of chipping away at the practical problems of XVA calculations, but let us focus on the burdens associated with the computation of sensitivities, for several reasons. First, sensitivities constitute a perennial problem in the quant world: whenever one computes some quantity, odds are that somebody in a trading or governance function will want to see sensitivities of said quantity to the inputs that are used in the computation, for limit monitoring, hedging, allocation, sanity checking, and so forth. Second, having input sensitivities available can be very powerful in an optimization setting. One rapidly growing area of “big finance” where optimization problems are especially pervasive is in the machine learning space, an area that is subject to enormous interest at the moment. And third, it just happens that there exists a very powerful technique to reduce the computational burden of sensitivity calculations to almost magically low levels.

To expand on the last point above, let us note the following quote by Phil Wolfe ([1]):

There is a common misconception that calculating a function of n variables and its gradient is about $n + 1$ times as expensive as just calculating the function. This will only be true if the gradient is evaluated by differencing function values or by some other emergency procedure. If care is taken in handling quantities, which are common to the function and its derivatives, the ratio is usually 1.5, not $n + 1$, whether the quantities are defined explicitly or implicitly, for example, the solutions of differential equations...

The “care” in “handling quantities” that Wolfe somewhat cryptically refers to is now known as *Algorithmic Adjoint Differentiation* (AAD), also known as *reverse automatic differentiation* or, in machine learning circles, as *backward propagation* (or simply *backprop*). Translated into our XVA example, the promise of the “cheap gradient” principle underpinning AAD is that computation of all sensitivities to the XVA metric – no matter how many thousands of sensitivities this might be – may be computed at a cost that is order $\mathcal{O}(1)$ times the cost of computing the basic XVA metric itself. It can be shown (see [2]) that the constant in the $\mathcal{O}(1)$ term is bounded from above by 5. To paraphrase [3], this remarkable result can be seen as somewhat of a “holy grail” of sensitivity computation.

The history of AAD is an interesting one, marked by numerous discoveries and re-discoveries of the same basic idea which, despite its profoundness,¹

¹Nick Trefethen [4] classifies AAD as one of the 30 greatest numerical algorithms of the 20th century.

has had a tendency of sliding into oblivion; see [3] for an entertaining and illuminating account. The first descriptions of AAD date back to the 1960s, if not earlier, but did not take firm hold in the computer science community before the late 1980s. In Finance, the first published account took another 20 years to arrive, in the form of the award-winning paper [5].

As one starts reading the literature, it soon becomes clear why AAD originally had a hard time getting a foothold: the technique is hard to comprehend; is often hidden behind thick computer science lingo or is buried inside applications that have little general interest.² Besides, even if one manages to understand the ideas behind the method, there are often formidable challenges in actually implementing AAD in code, especially with management of memory or retro-fitting AAD into an existing code library.

The book you hold in your hands addresses the above challenges of AAD head-on. Written by a long-time derivatives quant, Antoine Savine, the exposition is done at a level, and in an applications setting, that is ideal for a Finance audience. The conceptual, mathematical, and computational ideas behind AAD are patiently developed in a step-by-step manner, where the many brain-twisting aspects of AAD are demystified. For real-life application projects, the book is loaded with modern C++ code and battle-tested advice on how to get AAD to run *for real*.

Select topics include: parallel C++ programming, operator overloading, tapes, check-pointing, model calibration, and much more. For both newcomers and those quaint exotics quants among us who need an upgrade to our coding skills and to our understanding of AAD, my advice is this: start reading!



²Some of the early expositions of AAD took place in the frameworks of chemical engineering, electronic circuits, weather forecasting, and compiler optimization.