

Intel® oneAPI Math Kernel Library Cookbook

[Download PDF](#)[View More](#) ▾

A newer version of this document is available. Customers should [click here](#) to go to the newest version.

 Search this document

[Document
Table of
Contents](#)
[Document
Table of
Contents](#) X

[Intel® oneAPI
Math Kernel
Library
Cookbook](#)
[Intel® oneAPI
Math Kernel
Library
Cookbook](#) X

[Notices and
Disclaimers](#)
[Getting
Technical
Support](#)
[What's New](#)
[Notational
Conventions](#)
[Related
Information](#)
[Finding an
approximate
solution to a
stationary
nonlinear heat
equation](#)
[Factoring
general block
tridiagonal
matrices](#)

Using the Black-Scholes formula for European options pricing

Goal

Speed up Black-Scholes computation of European options pricing.

Solution

Use oneMKL vector math functions to speed up computation.

The Black-Scholes model describes the market behavior as a system of stochastic differential equations [Black73]. Call and put European options issued in this market are then priced according to the Black-Scholes formulae:

$$V_{\text{call}} = S_0 \cdot \text{CDF}(d_1) - e^{-rT} \cdot X \cdot \text{CDF}(d_2)$$

$$V_{\text{put}} = e^{-rT} \cdot X \cdot \text{CDF}(-d_2) - S_0 \cdot \text{CDF}(-d_1)$$

where

$$d_1 = \frac{\ln(S_0/X) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\ln(S_0/X) + (r - \sigma^2/2)T}{\sigma\sqrt{T}}$$

$V_{\text{call}}/V_{\text{put}}$ are the present values of the call/put options, S_0 is the present price of the stock, X is the strike price, r is the risk-neutral rate, σ is the volatility, T is the maturity and CDF is the cumulative distribution function of the standard normal distribution.

Alternatively, you can use the error function ERF which has a simple relationship with the cumulative normal distribution function:

$$\text{CDF}(x) = \frac{1}{2} + \frac{1}{2} \cdot \text{ERF}\left(\frac{x}{\sqrt{2}}\right)$$

Source code: see the `black-scholes` folder in the samples archive available at <https://software.intel.com/content/dam/develop/external/us/en/documents/mkl-cookbook-samples-120115.zip>.

Straightforward implementation of Black-Scholes

This code implements the closed form solution for pricing call and put options.

```
1 void BlackScholesFormula( int nopt,
2     tfloat r, tfloat sig, const tfloat s0[], const tfloat
3     x[],
4     const tfloat t[], tfloat vcall[], tfloat vput[] )
5 {
6     tfloat d1, d2;
7     int i;
8
9 }
```

Solving a system of linear equations with an LU-factored block tridiagonal coefficient matrix

Factoring block tridiagonal symmetric positive definite matrices

Solving a system of linear equations with a block tridiagonal symmetric positive definite coefficient matrix

Computing principal angles between two subspaces

Computing principal angles between invariant subspaces of block triangular matrices

Evaluating a Fourier integral

Using Fast Fourier Transforms for computer tomography image reconstruction

Noise filtering in financial market data streams

Using the Monte Carlo method for simulating

Using the Black-Scholes formula for European options pricing

```

10     for ( i=0; i<nopt; i++ )
11     {
12         d1 = ( LOG(s0[i]/x[i]) + (r + HALF*sig*sig)*t[i] )
13         /
14         ( sig*SQRT(t[i]) );
15         d2 = ( LOG(s0[i]/x[i]) + (r - HALF*sig*sig)*t[i] )
16         /
17         ( sig*SQRT(t[i]) );
18
19         vcall[i] = s0[i]*CDFNORM(d1) - EXP(

```

The number of options is specified as the `nopt` parameter. The `tfloat` type is either float or double depending on the precision you want to use. Similarly `LOG`, `EXP`, `SQRT`, and `CDFNORM` map to single or double precision versions of the respective math functions. The constant `HALF` is either 0.5f or 0.5 for single and double precision, respectively.

In addition to `nopt`, the input parameters for the algorithm are `s0` (present stock price), `r` (risk-neutral rate), `sig` (volatility), `t` (maturity), and `x` (strike price). The result is returned in `vcall` and `vput` (the present value of the call and put options, respectively).

It is assumed that `r` and `sig` are constant for all options being priced; the other parameters are arrays of floating-point values. The `vcall` and `vput` parameters are output arrays.

Discussion

Transcendental functions are at the core of the Black-Scholes formula benchmark. However, each option value depends on five parameters and as the math is computed faster, the memory effects become more pronounced. Thus the number of array parameters is an important factor that can change the computations from being compute-limited to memory bandwidth limited. Additionally, memory size constraints should be considered when pricing hundred millions of options.

The Intel C++ Compiler provides vectorization and parallelization controls that might help uncover the SIMD and multi-core potential of Intel Architecture with respect to the Black-Scholes formula. Optimized vectorized math functionality is available with the Short Vector Math Library (SVML) runtime library.

The oneMKL Vector Mathematical (VM) functions provide highly tuned transcendental math functionality that can be used to further speed up formula computations.

There are several opportunities for optimization of the straightforward code: hoisting common sub-expressions out of loops, replacing the `CDFNORM` function with the `ERF` function (which is usually faster), exploiting the relationship $\text{ERF}(-x) = -\text{ERF}(x)$, and replacing the division by `SQRT` with multiplication by the reciprocal square root `INVSQRT` (which is usually faster).

Optimized implementation of Black-Scholes

```

1 void BlackScholesFormula( int nopt,
2     tfloat r, tfloat sig, tfloat s0[], tfloat x[],
3     tfloat t[], tfloat vcall[], tfloat vput[] )
4 {
5     int i;
6     tfloat a, b, c, y, z, e;
7     tfloat d1, d2, w1, w2;
8     tfloat mr = -r;
9     tfloat sig_sig_two = sig * sig * TWO;
10
11     for ( i = 0; i < nopt; i++ )
12     {
13         a = LOG( s0[i] / x[i] );
14         b = t[i] * mr;
15         z = t[i] * sig_sig_two;
16
17         c = QUARTER * z;
18         e = EXP ( b );

```

[European options pricing](#)
[Using the Black-Scholes formula for European options pricing](#)
[Multiple simple random sampling without replacement](#)
[Using a histospline technique to scale images](#)
[Speeding up Python* scientific computations](#)
[Bibliography](#)


[Intel® oneAPI Math Kernel Library Cookbook](#)

- [Notices and Disclaimers](#)
- [Getting Technical Support](#)
- [What's New](#)
- [Notational Conventions](#)
- [Related Information](#)
- [Finding an approximate solution to a stationary nonlinear heat equation](#)
- [Factoring general block tridiagonal matrices](#)
- [Solving a system of linear equations with an LU-factored block tridiagonal coefficient matrix](#)
- [Factoring block tridiagonal symmetric](#)

Using the Black-Scholes formula for European options pricing

In this code `INVSQRT(x)` is either `1.0/sqrt(x)` or `1.0f/sqrtf(x)` depending on precision; `TWO` and `QUARTER` are the floating-point constants 2 and 0.25 respectively.

Discussion

A few optimizations help generating effective code using the Intel® C/C++ compiler. See the *User and Reference Guide for the Intel® C++ Compiler* for more details about the compiler pragmas and switches suggested in this section.

Apply `#pragma simd` to tell the compiler to vectorize the loop and `#pragma vector aligned` to notify the compiler that the arrays are aligned (you need to properly align vectors at the memory allocation stage) and that it is safe to rely on aligned load and store instructions. Efficient vectorization, such as that available with SVML, can achieve a speedup of several times versus scalar code.



```
1  ...
2  #pragma simd
3  #pragma vector aligned
4  for ( i = 0; i < nopt; i++ )
5  ...
```

With these changes the code can take advantage of all available CPU cores. The simplest way is to add the `-autopar` switch to the compilation line so that the compiler attempts to parallelize the code automatically. Another option is to use the standard OpenMP* pragma:



```
1  ...
2  #pragma omp parallel for
3  for ( i = 0; i < nopt; i++ )
4  ...
```

Further performance improvements are possible if you can relax the accuracy of math functions using Intel C++ Compiler options such as `-fp-model fast`, `-no-prec-div`, `-ftz`, `-fimf-precision`, `-fimf-max-error`, and `-fimf-domain-exclusion`.

NOTE:

Linux* OS specific syntax is given for the compiler switches. See the *User and Reference Guide for the Intel® C++ Compiler* for more detail.

In massively parallel cases, the compute time of math functions can be low enough for memory bandwidth to emerge as the limiting factor for loop performance. This can impair the otherwise linear speedup from parallelism. In such cases memory bandwidth friendly non-temporal load/store instructions can help:



```
1  ...
2  #pragma vector nontemporal
3  for ( i = 0; i < nopt; i++ )
4  ...
```

The oneMKL VM component provides highly tuned transcendental math functions that can help further improving performance. However, using them requires refactoring of the code to accommodate for the vector nature of the VM APIs. In the following code example, non-trivial math functions are taken from VM, while remaining basic arithmetic is left to the compiler.

A temporary buffer is allocated on the stack of the function to hold intermediate results of vector math computations. It is important for the buffer to be aligned on the maximum applicable SIMD register size. The buffer size is chosen to be

positive definite matrices

- Solving a system of linear equations with a block tridiagonal symmetric positive definite coefficient matrix
- Computing principal angles between two subspaces
- Computing principal angles between invariant subspaces of block triangular matrices
- Evaluating a Fourier integral
- Using Fast Fourier Transforms for computer tomography image reconstruction
- Noise filtering in financial market data streams
- Using the Monte Carlo method for simulating European options pricing
- Using the Black-Scholes formula for European options pricing
- Goal Solution Discussion Routines Used
- Multiple simple random sampling without replacement
- Using a histospline technique to scale images
- Speeding up Python* scientific computations
- Bibliography

Company Overview

Contact Intel

Newsroom

Using the Black-Scholes formula for European options pricing

large enough for the VM functions to achieve their best performance (compensating for vector function startup cost), yet small enough to maximize cache residence of the data. You can experiment with buffer size; a suggested starting point is NBUF=1024.

oneMKL VM implementation of Black-Scholes

```
1 void BlackScholesFormula_MKL( int nopt,
2     tfloat r, tfloat sig, tfloat * s0, tfloat * x,
3     tfloat * t, tfloat * vcall, tfloat * vput )
4 {
5     int i;
6     tfloat mr = -r;
7     tfloat sig_sig_two = sig * sig * TW0;
8
9     #pragma omp parallel for
10 \
11     shared(s0, x, t, vcall, vput, mr, sig_sig_two,
12 nopt) \
13     default(none)
14     for ( i = 0; i < nopt; i+= NBUF )
15     {
16         int j;
17         tfloat *a, *b, *c, *y, *z, *e;
18         tfloat *d1, *d2, *w1, *w2;
```

For comparable precisions, oneMKL VM can deliver 30-50% better performance versus Intel Compiler and SVML-based solutions if the problem is compute bound (the data fits in the L2 cache). In this case the latency of cache read/write operations is masked by computations. Once the memory bandwidth emerges as a factor with the growth of the problem size, it becomes more important to optimize the memory usage, and the Intel VM solution based on intermediate buffers can lose its advantage to the no-buffering one-pass solution with SVML.

Routines Used

Task	Routine
Sets a new accuracy mode for VM functions according to the mode parameter and stores the previous VM mode to oldmode.	vmSetMode
Performs element by element division of vector a by vector b	vsdiv/vddiv
Computes natural logarithm of vector elements.	vsln/vdln
Computes an inverse square root of vector elements.	vsinvsqrt/vdinvsqrt
Computes an exponential of vector elements.	vsexp/vdexp
Computes the error function value of vector elements.	vserf/vderf

Parent topic: [Intel® oneAPI Math Kernel Library Cookbook](#)

← [Using the Monte Carlo method for simulating European options pricing](#)

[Multiple simple random sampling without replacement](#) →

[Investors](#)[Careers](#)[Corporate Responsibility](#)[Inclusion](#)[Public Policy](#)[© Intel Corporation](#)[Terms of Use](#)[*Trademarks](#)[Cookies](#)[Privacy](#)[Supply Chain Transparency](#)[Site Map](#)[Recycling](#)

Intel technologies may require enabled hardware, software or service activation. // No product or component can be absolutely secure. // Your costs and results may vary. // Performance varies by use, configuration, and other factors. Learn more at [intel.com/performanceindex](https://www.intel.com/performanceindex). // See our complete legal [Notices and Disclaimers](#). // Intel is committed to respecting human rights and avoiding causing or contributing to adverse impacts on human rights. See Intel's [Global Human Rights Principles](#). Intel's products and software are intended only to be used in applications that do not cause or contribute to adverse impacts on human rights.

