Computing infinite range integrals of an arbitrary product of Bessel functions

Joris Van Deun   Ronald Cools

Department of Computer Science
K.U.Leuven

September 17, 2005

Outline

Introduction

Algorithm

Finite range integration

Infinite range approximation

Optimization

Implementation issues

Cost parameter t_{GJ}

Backward compatibility

Examples

The SIAM 100-Digit Challenge

- 10 problems in high-accuracy numerical computing
- Book by winning teams Bornemann, Laurie, Wagon and Waldvogel
- Appendix D. More Problems.

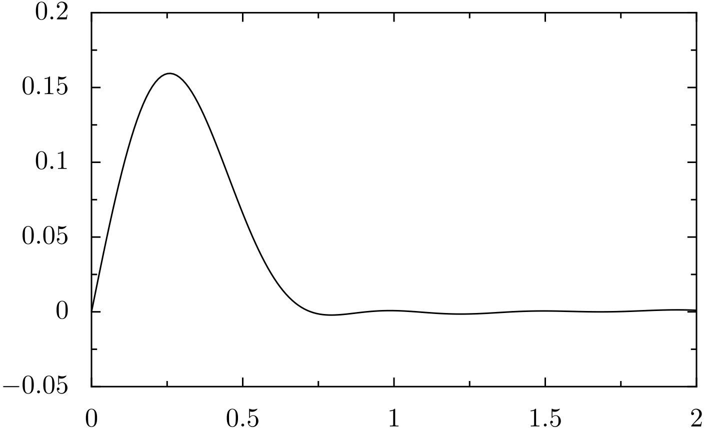
Problem 8

What is the value of

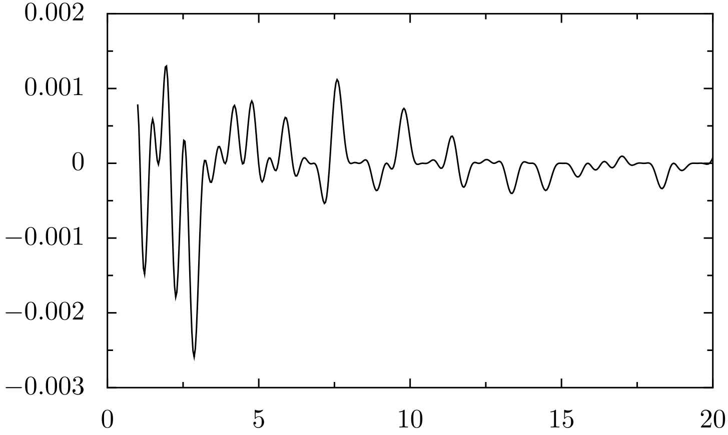
$$\int_0^\infty xJ_0(x\sqrt{2})J_0(x\sqrt{3})J_0(x\sqrt{5})J_0(x\sqrt{7})J_0(x\sqrt{11})dx,$$

where J_0 denotes the Bessel function of the first kind of order zero?

Integrand near 0



Integrand away from 0



Computational difficulties

- Infinite range
- Irregular oscillatory behaviour
- Slowly decaying integrand $\sim O(x^{-3/2})$

Possible solutions

- Extrapolation
- Double exponential formulas
- Integrate tail using incomplete Gamma function

Computational difficulties

- Infinite range
- Irregular oscillatory behaviour
- Slowly decaying integrand $\sim O(x^{-3/2})$

Possible solutions

- Extrapolation
- Double exponential formulas
- Integrate tail using incomplete Gamma function

More general setting

Compute the value of

$$I(\boldsymbol{a}, \boldsymbol{\nu}, m) = \int_0^\infty x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) dx$$

where

- $J_{\nu_i}(x)$ Bessel function of the first kind and order ν_i
- m real number such that $\sum_i \nu_i + m > -1$ (assures integrable singularity at 0)
- a_i strictly positive real numbers

Applications

Integrals of this kind occur in ...

- Calculation of products of nucleon propagators in a spherically symmetric medium
- Evaluation of water melon type Feynman diagrams
- Particle motion in an unbounded rotating fluid in magnetohydrodynamic flow
- Crack problems in elasticity
- Distortions of nearly circular lipid domains
- ...

Algorithm — basic idea

Split integral in finite and infinite part at breakpoint x_0

Finite part

$$I_1 = \int_0^{x_0} x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) dx$$

- low-order composite Gauss-Legendre
- extrapolation to 0 if algebraic singularity

Infinite part

$$I_2 = \int_{x_0}^{\infty} x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) dx$$

- asymptotic expansion for J_{ν_i}
- integrate using incomplete Gamma function

Algorithm — basic idea

Split integral in finite and infinite part at breakpoint x_0

Finite part

$$I_1 = \int_0^{x_0} x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) dx$$

- low-order composite Gauss-Legendre
- extrapolation to 0 if algebraic singularity

Infinite part

$$I_2 = \int_{x_0}^{\infty} x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) dx$$

- asymptotic expansion for J_{ν_i}
- integrate using incomplete Gamma function

Algorithm — basic idea

Split integral in finite and infinite part at breakpoint x_0

Finite part

$$I_1 = \int_0^{x_0} x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) dx$$

- low-order composite Gauss-Legendre
- extrapolation to 0 if algebraic singularity

Infinite part

$$I_2 = \int_{x_0}^{\infty} x^m \prod_{i=1}^{k} J_{\nu_i}(a_i x) dx$$

- asymptotic expansion for J_{ν_i}
- integrate using incomplete Gamma function

Finite part

- Split $[0, x_0]$ at equidistant points
- Number of subintervals $\sim$ estimated number of zeros in integrand using low-order approximation

$$J_\nu(x) \approx \sqrt{\frac{2}{\pi x}} \cos\left(x - \left(\frac{\nu}{2} + \frac{1}{4}\right)\pi\right).$$

- Hard-coded Gauss-Legendre rules on each subinterval until requested precision
- Average degree of quadrature rule to reach machine precision $\approx 15 + 19$ (error estimate included)

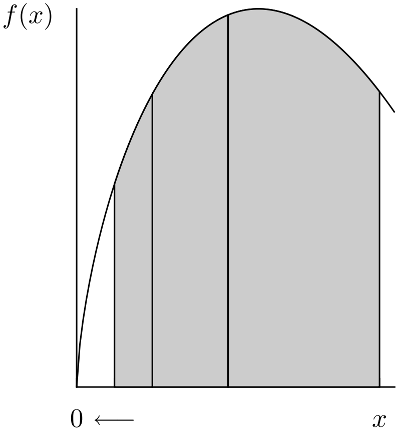
Finite part — extrapolation

Integrand $f(x)$ satisfies

$$f(x) = x^p \sum_{i=0}^{\infty} \alpha_i x^i, \quad x \to 0,$$

where $p = \sum_i \nu_i + m$.

- For non-integer p algebraic singularity in 0
- Extrapolate à la Richardson to remove singularity



Infinite part

There exist functions $P(\nu, x)$ and $Q(\nu, x)$ such that

$$J_\nu(x) = \sqrt{\frac{2}{\pi x}} \left[P(\nu, x) \cos \chi - Q(\nu, x) \sin \chi \right]$$

where $\chi = x - (\nu/2 + 1/4)\pi$.
P and Q admit known asymptotic expansions

$$P(\nu, x) \sim \sum_{j=0}^{\infty} c_{\nu,j} x^{-2j}$$

$$Q(\nu, x) \sim \sum_{j=0}^{\infty} d_{\nu,j} x^{-2j-1} \qquad x \longrightarrow \infty$$

Upper incomplete Gamma function

$$\Gamma(a,x) = \int_x^\infty t^{a-1} e^{-t} dt$$

- Generalization of Gamma function $\Gamma(a) = \Gamma(a,0)$
- Extended to arbitrary complex a and x by analytic continuation
- Efficient evaluation using Legendre's continued fraction expansion

$$\Gamma(a,x) = \frac{e^{-x}x^a}{x+} \frac{1-a}{1+} \frac{1}{x+} \frac{2-a}{1+} \frac{2}{x+} \cdots$$

- From the definition we obtain

$$\int_{x_0}^\infty e^{\mathbf{i}\alpha x} x^\beta dx = \left(\frac{\mathbf{i}}{\alpha}\right)^{\beta+1} \Gamma(\beta+1, -\mathbf{i}\alpha x_0)$$

Evaluating infinite range integrals using $\Gamma(a, x)$

Starting from asymptotic expansion for J_ν ...

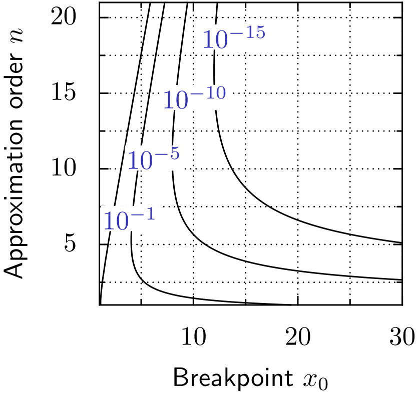
- Convert sine and cosine to exponentials
- Take $n+1$ terms in expansion
- Approximate tail of integrand

...leads to $2^{k-1}(k(2n+1)+1)$ integrals of the form

$$\int_{x_0}^{\infty} e^{\mathbf{i}\eta_i x} x^{m-k/2-j} dx, \quad i = 1, 2, \ldots, 2^{k-1}, \quad j = 0, 1, \ldots, k(2n+1)$$

where $\eta_i = a_1 \pm a_2 \pm \ldots \pm a_k$ (summing over all possible combinations).

Determining breakpoint x_0 and approximation order n



Contour plot of relative accuracy for infinite part $\delta(x_0, n)$

- *A priori* error estimates $\delta(x_0, n)$ and $\Delta(x_0, n)$ follow from error analysis
- Different (x_0, n) combinations lead to same accuracy
- Choose parameters to minimize computational effort

  $\rightarrow$ cost function

A suitable cost function

Major computational effort from evaluating Γ and J_ν

- On average $N = 15 + 19$ Bessel function evaluations in quadrature formula *per subinterval*
- Estimate number of subintervals from approximate number of zeros in integrand
- Incomplete Gamma function is called $2^{k-1}(k(2n+1)+1)$ times (at most)

Ignoring fixed cost gives cost function

$$\chi(x_0, n) = 2^k n t_{GJ} + \frac{x_0}{\pi} \frac{N}{2} \sum_{j=1}^{k} a_j$$

where t_{GJ} is relative efficiency of Γ compared to J_ν

Optimization problem

If relative error should not exceed ϵ

Find the values of x_0 and n which minize the cost function $\chi(x_0, n)$ with the constraint that $\delta(x_0, n) \leq \epsilon$

Using Lagrange multipliers leads to system of two nonlinear equations

$$\frac{\partial \delta}{\partial x_0} 2^k t_{GJ} = \frac{\partial \delta}{\partial n} \frac{N}{2\pi} \sum_{j=1}^{k} a_j \qquad (1)$$

$$\delta(x_0, n) = \epsilon$$

Approximate solution to (1) yields linear relation between x_0 and n

Nonlinear equation

Approximate optimal parameters (x_0, n) satisfy

$$x_0 = \frac{\kappa}{W(\kappa)} \left[n + \frac{1}{4} \left(3 - \frac{1}{1 + W(\kappa)} \right) \right]$$

where

$$\kappa = \frac{2^{k+1} t_{GJ} \pi}{N \sum_{j=1}^{k} a_j}.$$

and $W(x)$ is the Lambert W-function, $x = We^W$
Substituting in

$$\delta(x_0, n) = \epsilon$$

gives nonlinear equation in $n \rightarrow$ solve using Dekker-Brent

Matlab-programs

`igamma.m`

- Incomplete Gamma function
- Matlab's `gammainc.m` does not support complex arguments
- Fortran-to-Matlab conversion of program by Kostlan and Gokhman (1987)
- Small improvements

`besselint.m`

- Implementation of our algorithm
- Toolbox-independent
- Easy translation into other languages

Determining t_{GJ}

Definition

$$t_{GJ} = \frac{t_\Gamma}{t_J}$$

where

t_Γ average execution time for one call to `igamma`

t_J average execution time for one call to `besselj`

Problems

- Matlab supports vector operations $\longrightarrow$ average call to `besselj` has argument of size ≈ 17
- Cost function is not exact

gettGJ.m

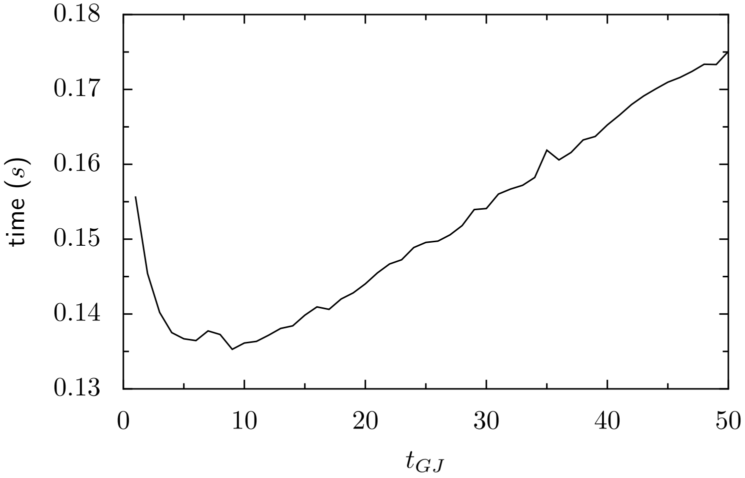
Implementation

- $2040 = 17 * 120$ calls to `igamma` (scalar argument) and `besselj` (vector argument)
- Partial loop unrolling to avoid zero timings on fast machines
- Averaging over 22 runs

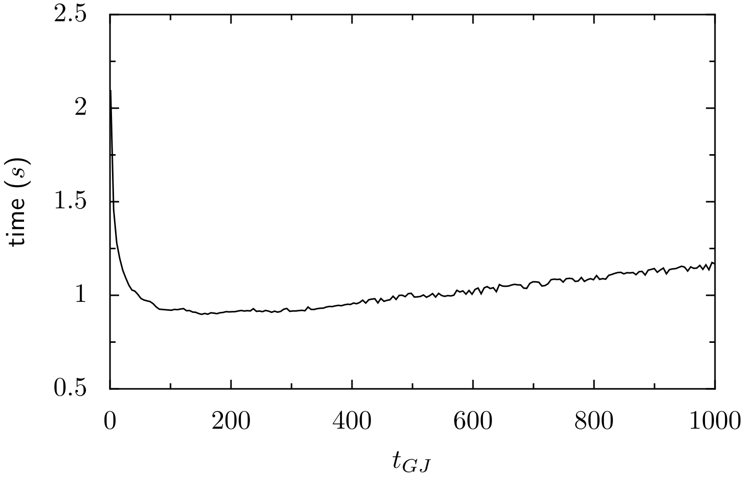
Results on Intel processor $2.80GHz$

	Matlab	Octave
t_{GJ}	19	1357

Execution time `besselint.m` vs. t_{GJ} — Matlab



Execution time `besselint.m` vs. t_{GJ} — Octave



Backward compatibility

Our code has been tested under

- Matlab 7.0.1.24704 (R14) Service Pack 1 for Linux,
- Matlab 6.5.0.180913a (R13) for Linux and Windows
- Matlab 6.1.0.450 (R12.1) for Windows
- Octave 2.1.69

Backward compatibility issues have negative impact on efficiency

Logical operators

Matlab ≥6.5 and Octave

	Short-circuit	Element-wise
and	&&	&
or	||	|

Matlab <6.5

- Only & and |
- Automatic short-circuit in condition after if-statement

Problem

- Because of backward compatibility we use & and |
- Makes `igamma` two times slower in Matlab 7.0

Numerical examples

Syntax for `besselint`

```
[f,err] = besselint(a,nu,m,reltol,abstol)
```

where

- `f` result
- `err` (optional) absolute and relative error estimates
- `a` vector with coefficients a_j
- `nu` vector with orders ν_j
- `m` power of x
- `reltol` (optional) relative tolerance
- `abstol` (optional) absolute tolerance

Excerpt from `experiments.m` with added timings

```
Part I: explicitly known examples

Ex. 1.2: a=[1]; nu=[-1/4]; m=1/3;
Exact answer: 4.699242939646014e-01
f =
     4.699242939646101e-01
time =
     9.648300000000098e-02
```

```
Ex. 2.1: a=[1 5]; nu=[0 1]; m=0;
Exact answer: 1/5
f =
     2.000000000000000e-01
time =
     1.373540000000002e-01

Ex. 4.1: a=sqrt([2 3 5 7]); nu=0; m=1;
Exact answer: 1.104110282210471e-01
f =
     1.104110282210470e-01
time =
     2.150379999999998e-01
```

```
Part II: how reliable are the error estimates?

Ex. 7.1: a=sqrt([2 3 5]); nu=0; m=1
Requested relative precision: 1e-14
Estimated relative precision: 2.35e-15
Actual relative precision: 1.67e-15

Ex. 8.2: a=sqrt([2 3 5 7]); nu=0; m=1
Requested absolute precision: 1e-6
Estimated absolute precision: 8.04e-07
Actual absolute precision: 4.11e-09
```

```
Part III: some unusual cases

Ex. 9.1: a=[1 1]; nu=0;  m=1;
Exact answer: -Inf
Assuming a(1)-a(2)=0;
Warning: Possible discontinuous case;
 result and error estimate may be inaccurate
> In besselint at 106
f =
  -Inf
time =
     1.103760000000005e-01
```

```
Ex. 11.1: a=[1 10 1000]; nu=0; m=1;
Exact answer: 0
f =
   -8.655453817121351e-17
time =
     4.545179000000001e+00

Ex. 11.3: a=1; nu=100; m=0;
Exact answer: 1
f =
     9.999999999989873e-01
time =
     8.703063000000000e+00
```