Astro 585: HW 4

Codename: The Maxwell-Jüttner Distribution

February 21, 2014

My git repository is here: `https://github.com/hsgg/astro585`, clone URL `https://github.com/hsgg/astro585.git`.

1 Profiling

```
#!/usr/bin/env julia

# Calculate a normalized Gaussian. Write it in this odd way, so we can see the
# profiler output better
function log_Gaussian(z, y, sigma)
    tmp1 = y - z
    tmp1 ./= sigma
    tmp1 = tmp1.^2
    tmp1 .*= -0.5
    tmp2 = sqrt(2 * pi)
    tmp2 .*= sigma
    tmp2 = log(tmp2)
    tmp1 -= tmp2
    return tmp1
end

function log_likelihood(log_pdf, y::Array, sigma::Array, z::Array)
    n = length(y)
    @assert n == length(sigma)
    @assert n == length(z)
    sum = zero(y[1])
    for i in 1:n
        a = y[i]
        b = z[i]
        c = sigma[i]
        tmp = log_pdf(a, b, c)
        sum += tmp
    end
    return sum
end
```

```
# make data:
Nobs = 1000
srand(42)
z = zeros(Nobs)
sigma = 2. * ones(Nobs)
y = z + sigma .* randn(Nobs)

# profile:
Profile.clear()
t = @elapsed @profile (for i in 1:10^4;
        log_likelihood(log_Gaussian, y, sigma, z);
    end)
println("Time elapsed: ", t, " seconds")
Profile.print()
println()

# I expect the function calls to take the most time, and so that's where the
# profile is most often, 5800 times calling the function "log_Gaussian()", and
# 1715 times 'log()':

# julia> include("2_profile.jl")
# Time elapsed: 8.564663583 seconds
# 7690 boot.jl; include; line: 238
#     7690 util.jl; anonymous; line: 42
#       20   ...mp/hw4/2_profile.jl; log_likelihood; line: 21
#       15   ...mp/hw4/2_profile.jl; log_likelihood; line: 22
#       172  ...mp/hw4/2_profile.jl; log_likelihood; line: 23
#       16   ...mp/hw4/2_profile.jl; log_likelihood; line: 24
#       36   ...mp/hw4/2_profile.jl; log_likelihood; line: 25
#       5800 ...mp/hw4/2_profile.jl; log_likelihood; line: 26
#         17   ...mp/hw4/2_profile.jl; log_Gaussian; line: 11
#          13   ...p/hw4/2_profile.jl; log_Gaussian; line: 7
#          32   ...p/hw4/2_profile.jl; log_Gaussian; line: 11
#          1715 ...p/hw4/2_profile.jl; log_Gaussian; line: 12
#          18   ...p/hw4/2_profile.jl; log_Gaussian; line: 14
#            3    inference.jl; typeinf_ext; line: 1092
#               1 inference.jl; typeinf; line: 1196
#               1 inference.jl; typeinf; line: 1382
#                 1 inference.jl; inlining_pass; line: 1956
#                   1 inference.jl; inlining_pass; line: 1972
#                     1 inference.jl; inlining_pass; line: 2014
#                       1 inference.jl; inlineable; line: 1832
#               1 inference.jl; typeinf; line: 1385
#                1 inference.jl; tuple_elim_pass; line: 2244
```

```
#                1 inference.jl; find_sa_vars; line: 2181
#       1589 ...mp/hw4/2_profile.jl; log_likelihood; line: 27
#         13   float.jl; +; line: 132
#          8  inference.jl; typeinf_ext; line: 1092
#             4 inference.jl; typeinf; line: 1259
#               2 inference.jl; abstract_interpret; line: 958
#                 2 inference.jl; abstract_eval; line: 814
#                  2 inference.jl; abstract_eval_call; line: 789
#                    2 inference.jl; abstract_call; line: 701
#                      2 inference.jl; abstract_call_gf; line: 576
#                         2 reflection.jl; _methods; line: 77
#                            2 reflection.jl; _methods; line: 97
#                               2 reflection.jl; _methods; line: 97
#                                  2 reflection.jl; _methods; line: 80
#               2 inference.jl; abstract_interpret; line: 966
#                1 inference.jl; abstract_eval; line: 814
#                 1 inference.jl; abstract_eval_call; line: 788
#                 1 inference.jl; abstract_eval; line: 814
#                  1 inference.jl; abstract_eval_call; line: 756
#                   1 array.jl; getindex; line: 296
#                    1 array.jl; copy!; line: 51
#                     1 array.jl; unsafe_copy!; line: 37
#             1 inference.jl; typeinf; line: 1379
#               1 inference.jl; type_annotate; line: 1508
#                 1 inference.jl; eval_annotate; line: 1471
#                   1 inference.jl; eval_annotate; line: 1483
#             3 inference.jl; typeinf; line: 1382
#               3 inference.jl; inlining_pass; line: 1956
#                 2 inference.jl; inlining_pass; line: 1972
#                   1 inference.jl; inlining_pass; line: 1990
#                     1 inference.jl; exprtype; line: 1662
#                       1 inference.jl; abstract_eval; line: 798
#                         1 inference.jl; abstract_eval_symbol; line: 910
#                          1 inference.jl; abstract_eval_global; line: 898
#                            1 inference.jl; abstract_eval_constant; line: 875
#                   1 inference.jl; inlining_pass; line: 2014
#                     1 inference.jl; inlineable; line: 1772
#                        1 reflection.jl; _methods; line: 77
#                           1 reflection.jl; _methods; line: 97
#                              1 reflection.jl; _methods; line: 97
#                                 1 reflection.jl; _methods; line: 80
#                 1 inference.jl; inlining_pass; line: 2014
#                   1 inference.jl; inlineable; line: 1910
#                     1 inference.jl; sym_replace; line: 1560
#                       1 inference.jl; sym_replace; line: 1560
```

```
#                          1 inference.jl; sym_replace; line: 1560
# 8    float.jl; +; line: 132


# Unfortunately, the the julia does not appear to inline the call to
# 'log_Gaussian()'. This would be nice. Indeed, inlineing it by hand results in
# a 5x speedup.
#
# Another way to speed it up would be to not loop in 'log_likelihood()', but
# use vectors in 'log_Gaussian()'. This might work, and also reduce the number
# of function calls to 'log()', but it could lead to many spurious memory
# allocations.
#
# Let's try that:
function log_likelihood_vec(log_pdf, y::Array, sigma::Array, z::Array)
    n = length(y)
    @assert n == length(sigma)
    @assert n == length(z)
    s = log_pdf(y, z, sigma)
    su = sum(s)
    return su
end

Profile.clear()
t = @elapsed @profile (for i in 1:10^4;
        log_likelihood_vec(log_Gaussian, y, sigma, z);
    end)
println("Time elapsed: ", t, " seconds")
Profile.print()

# Nice, more thatn 2 times increase in speed:

# Time elapsed: 3.672488957 seconds
# 3205 boot.jl; include; line: 238
#     3205 util.jl; anonymous; line: 72
#       3143 ...mp/hw4/2_profile.jl; log_likelihood_vec; line: 65
#         97   ...mp/hw4/2_profile.jl; log_Gaussian; line: 6
#          73 array.jl; -; line: 135
#          3  array.jl; -; line: 136
#          8  array.jl; -; line: 926
#          13 array.jl; -; line: 927
#         1121 ...mp/hw4/2_profile.jl; log_Gaussian; line: 7
#          901 broadcast.jl; ##broadcast_T_/#233; line: 187
#             4   broadcast.jl; broadcast_shape; line: 27
#               2 broadcast.jl; longer_size; line: 21
```

```
#                1 broadcast.jl; longer_size; line: 20
#             6   broadcast.jl; broadcast_shape; line: 29
#              3 array.jl; fill!; line: 183
#             8   broadcast.jl; broadcast_shape; line: 42
#          14  broadcast.jl; ##broadcast_T_/#233; line: 188
#          204 broadcast.jl; ##broadcast_T_/#233; line: 189
#            13  broadcast.jl; broadcast_args; line: 87
#             1 array.jl; sizehint; line: 709
#             5 broadcast.jl; calc_loop_strides; line: 63
#             2 broadcast.jl; calc_loop_strides; line: 64
#             1 broadcast.jl; calc_loop_strides; line: 66
#              1 array.jl; sizehint; line: 709
#             2 broadcast.jl; calc_loop_strides; line: 77
#             2 broadcast.jl; calc_loop_strides; line: 83
#            3   broadcast.jl; broadcast_args; line: 88
#              10  broadcast.jl; ##/_inner!#235; line: 129
#               1 tuple.jl; ==; line: 81
#               6 tuple.jl; ==; line: 84
#                 1 promotion.jl; ==; line: 188
#              1   broadcast.jl; ##/_inner!#235; line: 131
#              27  broadcast.jl; ##/_inner!#235; line: 136
#              138 broadcast.jl; ##/_inner!#235; line: 166
#         573  ...mp/hw4/2_profile.jl; log_Gaussian; line: 8
#          572 array.jl; .^; line: 920
#         100  ...mp/hw4/2_profile.jl; log_Gaussian; line: 9
#          71 array.jl; .*; line: 135
#          9  array.jl; .*; line: 136
#          5  array.jl; .*; line: 944
#          15 array.jl; .*; line: 945
#         1    ...mp/hw4/2_profile.jl; log_Gaussian; line: 10
#         92   ...mp/hw4/2_profile.jl; log_Gaussian; line: 11
#          60 array.jl; .*; line: 135
#          12 array.jl; .*; line: 136
#          8  array.jl; .*; line: 937
#          11 array.jl; .*; line: 938
#          1  array.jl; .*; line: 941
#         1058 ...mp/hw4/2_profile.jl; log_Gaussian; line: 12
#          1058 operators.jl; log; line: 236
#         88   ...mp/hw4/2_profile.jl; log_Gaussian; line: 13
#          57 array.jl; -; line: 135
#          3  array.jl; -; line: 136
#          18 array.jl; -; line: 926
#          10 array.jl; -; line: 927
#            3 inference.jl; typeinf_ext; line: 1092
#               1 inference.jl; typeinf; line: 1169
```

```
#               2 inference.jl; typeinf; line: 1382
#                 2 inference.jl; inlining_pass; line: 1956
#                   2 inference.jl; inlining_pass; line: 1972
#                     2 inference.jl; inlining_pass; line: 2014
#                       1 inference.jl; inlineable; line: 1808
#                       1 inference.jl; inlineable; line: 1908
#                         1 inference.jl; resolve_globals; line: 1624
#                           1 inference.jl; resolve_globals; line: 1624
#                             1 inference.jl; resolve_globals; line: 1624
#       44   ...mp/hw4/2_profile.jl; log_likelihood_vec; line: 66
#          43 abstractarray.jl; sum; line: 1487
#           43 abstractarray.jl; sum_pairwise; line: 1481
#            43 abstractarray.jl; sum_pairwise; line: 1481
#             43 abstractarray.jl; sum_pairwise; line: 1481
#              24 abstractarray.jl; sum_pairwise; line: 135
#              17 abstractarray.jl; sum_pairwise; line: 1475
#              2  abstractarray.jl; sum_pairwise; line: 1478
#          3  inference.jl; typeinf_ext; line: 1092
#             1 inference.jl; typeinf; line: 1259
#               1 inference.jl; abstract_interpret; line: 966
#                1 inference.jl; abstract_eval; line: 814
#                 1 inference.jl; abstract_eval_call; line: 788
#             2 inference.jl; typeinf; line: 1382
#               2 inference.jl; inlining_pass; line: 1956
#                 1 inference.jl; inlining_pass; line: 1943
#                  1 inference.jl; is_known_call; line: 2120
#                 1 inference.jl; inlining_pass; line: 1994
# 1    range.jl; colon; line: 36
# 1    range.jl; colon; line: 38
```

2 Loops, not Loops

```
#!/usr/bin/env julia

using Base.Test
using Devectorize

gaussian(x) = exp(-0.5 * x.^2) / sqrt(2 * pi)


# integrate_loop():
#    - Calculates the integal from 'a' to 'b' of the function 'fn(x)', using no
#      more than 'maxevals' function evaluations.
#    - Returns a pair (I, E) similar to 'quadgk()', where I is the integral,
#      and E an estimation of the error.
```

```
function integrate_loop(fn, a, b; maxevals=10^7)
    @assert maxevals >= 0
    N = maxevals
    sum = 0.0
    dx = (b-a)/(N+1)
    for i in 1:N
        sum += fn(a + i * dx)
    end
    I = sum * (b-a) / N

    # no idea if this is right, but it is in the right ballpark for the tests
    E = I / N

    return I, E
end


function integrate_vector(fn, a, b; maxevals=10^7)
    @assert maxevals >= 0
    N = maxevals
    x = linspace(a, b, N)
    y = gaussian(x)
    I = sum(y) * (b-a) / N
    E = I / N
    return I, E
end


function integrate_mapthenreduce(fn, a, b; maxevals=10^7)
    @assert maxevals >= 0
    N = maxevals
    x = linspace(a, b, N)
    y = map(gaussian, x)
    I = reduce(+, y) * (b-a) / N
    E = I / N
    return I, E
end


function integrate_mapreduce(fn, a, b; maxevals=10^7)
    @assert maxevals >= 0
    N = maxevals
    x = linspace(a, b, N)
    s = mapreduce(gaussian, +, x)
    I = s * (b-a) / N
```

```
    E = I / N
    return I, E
end


function integrate_devectorize(fn, a, b; maxevals=10^7)
    # Hm, to install Devectorize package, julia wants to use the
    # --single-branch option to git clone, which git learned in version 1.7.10.
    # The Davey computers only have 1.7.9. We are almost there!
    # Doing this on my laptop now...
    @assert maxevals >= 0
    N = maxevals
    x = linspace(a, b, N)
    @devec s = sum(exp(-0.5 .* x.^2) ./ sqrt(2 .* pi))
    I = s * (b-a) / N
    E = I / N
    return I, E
end


function test_integral(intfn; maxevals=10^7)
    a = 0.5
    numerically, ntol = intfn(gaussian, -a, a, maxevals=maxevals)
    analytically = erf(a/sqrt(2))
    @test_approx_eq_eps numerically analytically ntol
    if maxevals >= 10^3
        @test_approx_eq_eps ntol 0.0 1e-4
    elseif maxevals >= 10^6
        @test_approx_eq_eps ntol 0.0 1e-8
    end
end


function test_integral_function(intfn)
    # test at least one odd case
    @test_throws intfn(-0.3, 0.4, maxevals=-1)

    # test the results
    test_integral(intfn, maxevals=0)
    test_integral(intfn, maxevals=1)
    test_integral(intfn, maxevals=2)
    test_integral(intfn, maxevals=3)
    test_integral(intfn, maxevals=10^4)
    test_integral(intfn, maxevals=10^6)
end
```

```
function time_integral_func(intfn, integrandfn;
        maxcalls=10^2, maxevals=10^4)
    a = -0.3
    b = 0.6
    time = @elapsed for i in 1:maxcalls
        numerically, tol = intfn(integrandfn, a, b, maxevals=maxevals)
    end
    return time
end

function run_timing_test(name, intfn, integrandfn)
    time1 = time_integral_func(intfn, integrandfn, maxcalls=10^2, maxevals=10^4)
    time2 = time_integral_func(intfn, integrandfn, maxcalls=10^4, maxevals=10^2)
    s = ^(" ", 13 - length(name)) # 13 is the longest name
    println(name, ": ", s, time1, " sec each heavy,\t",
                time2, " sec called often")
end


println("Run tests...")
test_integral_function(integrate_loop)
test_integral_function(quadgk)
test_integral_function(integrate_vector)
test_integral_function(integrate_mapthenreduce)
test_integral_function(integrate_mapreduce)
test_integral_function(integrate_devectorize)
println("All tests passed.")


run_timing_test("loop", integrate_loop, gaussian)
run_timing_test("quadgk", quadgk, gaussian)
run_timing_test("vector", integrate_vector, gaussian)
run_timing_test("mapthenreduce", integrate_mapthenreduce, gaussian)
run_timing_test("mapreduce", integrate_mapreduce, gaussian)
run_timing_test("devectorize", integrate_devectorize, gaussian)


# Result before optimizing anything:
#
#   Testing...
#   All tests passed.
#   loop:          0.484841521 sec each heavy,0.603287741 sec called often
#   quadgk:        0.002404426 sec each heavy,0.300752033 sec called often
#   vector:        0.335771084 sec each heavy,0.350315113 sec called often
```

```
#   mapthenreduce: 0.904367259 sec each heavy,0.917444621 sec called often
#   mapreduce:     0.703033884 sec each heavy,0.655952055 sec called often
#   devectorize:   0.169991816 sec each heavy,0.249352334 sec called often
#
# Most functions are a little slower when called often as opposed to having
# long loops in them. However, the mapreduce version is slightly faster. The
# comparison with 'quadgk()' is unfair, as it terminates whenever the desired
# tolerance is reached (and it's a different algorithm).

# Profiling devectorize version:
println()
Profile.clear()
@profile time_integral_func(integrate_devectorize, gaussian, maxcalls=10^1, maxevals=10^5)
Profile.print()
#
# 170 boot.jl; include; line: 238
#     170 profile.jl; anonymous; line: 14
#           170 ...4/3_integration.jl; time_integral_func; line: 110
#                 80 ...3_integration.jl; integrate_devectorize; line: 71
#                  80 array.jl; linspace; line: 238
#                 90 ...3_integration.jl; integrate_devectorize; line: 72
#
# Lines 71 and 72 take most of the time. 71 just allocates memory: it is the
# call to 'linspace()'. Line 72 performs the loop. I don't see much potential
# for optimizing the loop, but getting rid of the allocation via 'linspace()'
# would make sense if the function gets called often.

# Profiling vector version:
println()
Profile.clear()
@profile time_integral_func(integrate_vector, gaussian, maxcalls=10^1, maxevals=10^5)
Profile.print()
#
# 306 boot.jl; include; line: 238
#     306 profile.jl; anonymous; line: 14
#           306 ...4/3_integration.jl; time_integral_func; line: 110
#                 82  ...3_integration.jl; integrate_vector; line: 34
#                  1  array.jl; linspace; line: 237
#                  81 array.jl; linspace; line: 238
#                 218 ...3_integration.jl; integrate_vector; line: 35
#                  218 ...3_integration.jl; gaussian; line: 6
#                   9   array.jl; .*; line: 135
#                   1   array.jl; .*; line: 136
#                   1   array.jl; .*; line: 938
#                   41  array.jl; ./; line: 135
```

```
#                    3   array.jl; ./; line: 136
#                    1   array.jl; ./; line: 945
#                    45  array.jl; .^; line: 920
#                    117 operators.jl; exp; line: 236
#                   6   ...3_integration.jl; integrate_vector; line: 36
#                    6 abstractarray.jl; sum; line: 1487
#                     6 abstractarray.jl; sum_pairwise; line: 1481
#                      6 abstractarray.jl; sum_pairwise; line: 1481
#                       6 abstractarray.jl; sum_pairwise; line: 1481
#                        6 abstractarray.jl; sum_pairwise; line: 1481
#                         6 abstractarray.jl; sum_pairwise; line: 1481
#                          6 abstractarray.jl; sum_pairwise; line: 1481
#                           6 abstractarray.jl; sum_pairwise; line: 1481
#                            1 ...ractarray.jl; sum_pairwise; line: 1480
#                            5 ...ractarray.jl; sum_pairwise; line: 1481
#                             5 ...ractarray.jl; sum_pairwise; line: 1481
#                              5 ...actarray.jl; sum_pairwise; line: 1481
#                               4 ...actarray.jl; sum_pairwise; line: 135
#                               1 ...actarray.jl; sum_pairwise; line: 1475
#
# Some of the time is spent in 'linspace()' again, but much more time is spent
# in 'gaussian()'. This is likely due to there being some temporary arrays
# copied for intermediate results.
```

3 Triad twists

```
#!/usr/bin/env julia

# I predict that the last twisted one would be fastest as it avoids branches
# ('abs()' better be implemented branch-less). Twist1 is likely the slowest, as
# the branch could go either way, with little predicting being possible. I
# don't expect too much variability depending on the vector size, except for
# twist2, as it needs to loop through them twice, and so would be much slower
# for large sizes.
#
# For 50% positive/negative: Little prediction possible, twist3 fastest.
# For all positive: Branch prediction should work well, twist2 slowest, not
# much difference otherwise.
# For all negative: The same as all positive.
# For 90% positive: Twist2 probably wouldn't be too bad, but not the fastest
# either.
#
# I predict twist3 to be the best in almost all situations, provided that
# 'abs()' does not incur a function call, and that it is implemented without
# branches.
```

```
function triad(b::Vector, c::Vector, d::Vector)
    assert(length(b)==length(c)==length(d))
    a = similar(b)
    for i in 1:length(a)
        a[i] = b[i] + c[i] * d[i]
    end
    return a
end

function triad_twist1(b::Vector, c::Vector, d::Vector)
    assert(length(b)==length(c)==length(d))
    a = similar(b)
    for i in 1:length(a)
        if c[i]<0.
            a[i] = b[i] - c[i] * d[i]
        else
            a[i] = b[i] + c[i] * d[i]
        end
    end
    return a
end

function triad_twist2(b::Vector, c::Vector, d::Vector)
    assert(length(b)==length(c)==length(d))
    a = similar(b)
    for i in 1:length(a)
        if c[i]<0.
            a[i] = b[i] - c[i] * d[i]
        end
    end
    for i in 1:length(a)
        if c[i]>0.
            a[i] = b[i] + c[i] * d[i]
        end
    end
    return a
end

function triad_twist3(b::Vector, c::Vector, d::Vector)
    assert(length(b)==length(c)==length(d))
    a = similar(b)
    for i in 1:length(a)
        cc = abs(c[i])
        a[i] = b[i] + cc * d[i]
```

```
    end
    return a
end


###### benchmark stuff
function make_data(Nobs::Int)
    srand(4242424242)
    b = randn(Nobs)
    c = randn(Nobs)
    d = randn(Nobs)
    return b, c, d
end

function time_func(name, fn, b, c, d)
    t = @elapsed fn(b, c, d)
    s = ^(" ", 13 - length(name)) # 13 is the longest name
    println(name, ": ", s, t, " sec")
end


data = make_data(10^7)

time_func("triad", triad, data...)
time_func("triad_twist1", triad_twist1, data...)
time_func("triad_twist2", triad_twist2, data...)
time_func("triad_twist3", triad_twist3, data...)

# triad:         0.28133741 sec
# triad_twist1:  0.413410574 sec
# triad_twist2:  0.635552881 sec
# triad_twist3:  0.283048662 sec
#
# ...as predicted.


###### profile stuff
function profile_func(name, fn, b, c, d)
    Profile.clear()
    @profile fn(b, c, d)
    println()
    println(name, ":")
    Profile.print()
end
```

```
profile_func("triad", triad, data...)
profile_func("triad_twist1", triad_twist1, data...)
profile_func("triad_twist2", triad_twist2, data...)
profile_func("triad_twist3", triad_twist3, data...)

# triad:
# 263 boot.jl; include; line: 238
#          263 ...comp/hw4/3_triad.jl; profile_func; line: 14
#            35  ...omp/hw4/3_triad.jl; triad; line: 24
#            228 ...omp/hw4/3_triad.jl; triad; line: 25
#
# triad_twist1:
# 380 boot.jl; include; line: 238
#          380 ...comp/hw4/3_triad.jl; profile_func; line: 14
#            25  ...omp/hw4/3_triad.jl; triad_twist1; line: 33
#            20  ...omp/hw4/3_triad.jl; triad_twist1; line: 34
#            111 ...omp/hw4/3_triad.jl; triad_twist1; line: 35
#            224 ...omp/hw4/3_triad.jl; triad_twist1; line: 37
#
# triad_twist2:
# 594 boot.jl; include; line: 238
#          594 ...comp/hw4/3_triad.jl; profile_func; line: 14
#            55  ...omp/hw4/3_triad.jl; triad_twist2; line: 46
#            47  ...omp/hw4/3_triad.jl; triad_twist2; line: 47
#            225 ...omp/hw4/3_triad.jl; triad_twist2; line: 48
#            66  ...omp/hw4/3_triad.jl; triad_twist2; line: 51
#            52  ...omp/hw4/3_triad.jl; triad_twist2; line: 52
#            136 ...omp/hw4/3_triad.jl; triad_twist2; line: 53
#            13  ...omp/hw4/3_triad.jl; triad_twist2; line: 56
#
# triad_twist3:
# 263 boot.jl; include; line: 238
#          263 ...comp/hw4/3_triad.jl; profile_func; line: 14
#            23  ...omp/hw4/3_triad.jl; triad_twist3; line: 62
#            28  ...omp/hw4/3_triad.jl; triad_twist3; line: 63
#            212 ...omp/hw4/3_triad.jl; triad_twist3; line: 64
#
# The computation line generally is hit the same number of times for each,
# except for the twist1 and twist2 implementations, where the sum of the two
# computational lines is much greater. This could be explained by the fact that
# branch mis-prediction by the CPU could have the processor spend time in that
# line even when it is not being executed. That's the best explanation I have
# right now.
```