

# Performance Testing

In this live script we test the performance of several of the algorithms.

## Comparing the performance of the D&C Partial Spectral Factorization Algorithm to the TQR Golub-Welsch Algorithm for computing a Gauss-Legendre Rule

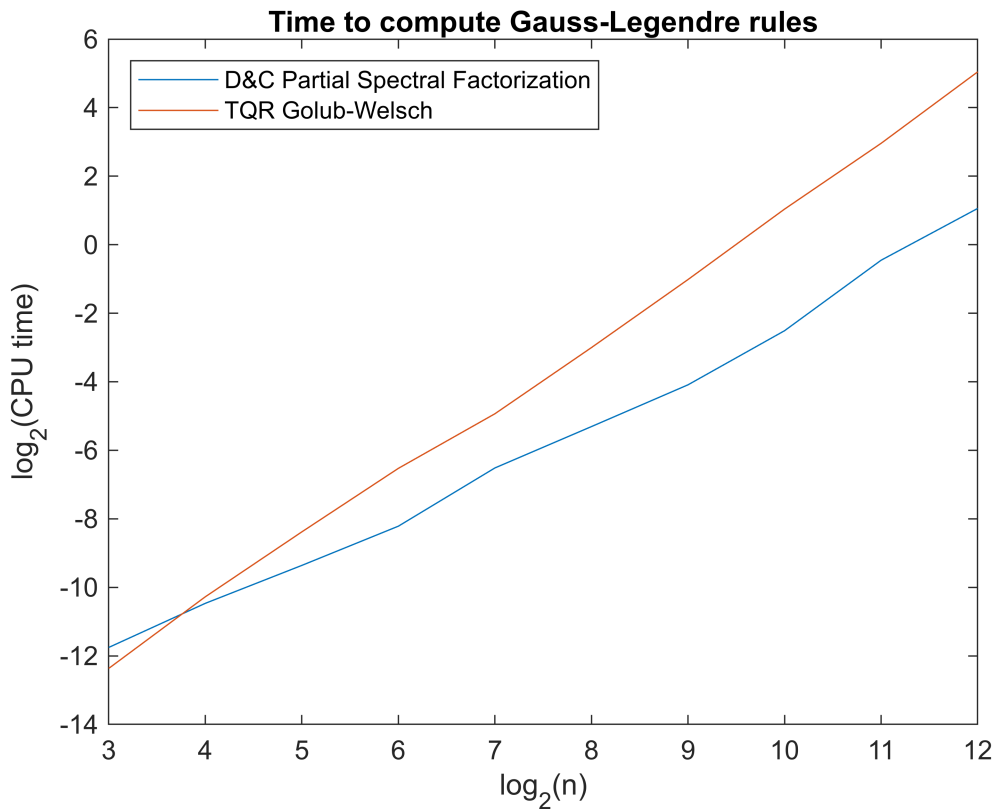
In this section we compare the performance of the D&C Partial Spectral Factorization Algorithm against the TQR Golub-Welsch Algorithm over a range of rule sizes when generating Gauss-Legendre rules. We will do this in a fairly crude manner using the Matlab `timeit()` function which we will run over a variety of problem sizes and then generate a plot of the times required by the two algorithms as a function of problem size.

```
% Loop over different problem sizes.
MAX = 12; % This takes a while if you choose a MAX > 11.
logn = 3:MAX;
tqr_times = zeros(size(logn));
dandc_times = zeros(size(logn));

for k=logn
    n = 2^k;
    % Generate the recurrence coefficients.
    [alpha, beta] = JacobiRecurrence(n,1,0);
    % Create function handles to the two algorithms.
    tqr = @() GaussRule(alpha,beta,'tqrGW'); % GaussRule() using the TQR Golub-
Welsch code.
    dandc = @() GaussRule(alpha,beta,'dandcPSF'); % GaussRule() using the Partial
Spectral Factorization D&C code.

    tqr_times(k) = timeit(tqr);
    dandc_times(k) = timeit(dandc);
end

plot((1:MAX),log2(dandc_times)); hold on;
plot((1:MAX),log2(tqr_times));
xlabel('log_{2}(n)');
ylabel('log_{2}(CPU time)');
title('Time to compute Gauss-Legendre rules');
legend('D&C Partial Spectral Factorization','TQR Golub-
Welsch','location','northwest');
hold off
```



## Comparing the Performance of the Nested D&C GaussPlusAntiGauss code vs. the Traditional Approach

This section demonstrates the performance of the Nested D&C GaussPlusAntiGauss code by using it to compute an  $n$ -point Gauss-Legendre and the  $n+1$  point Anti-Gauss rule. It then compares this to a traditional approach where the  $n$ -point Gauss-Legendre rule and the  $n+1$  point Anti-Gauss rule are both computed with independent applications of the TQR Golub-Welsch algorithm.

```
logn = 3:MAX;
nested_times = zeros(size(logn));
trad_times = zeros(size(logn));

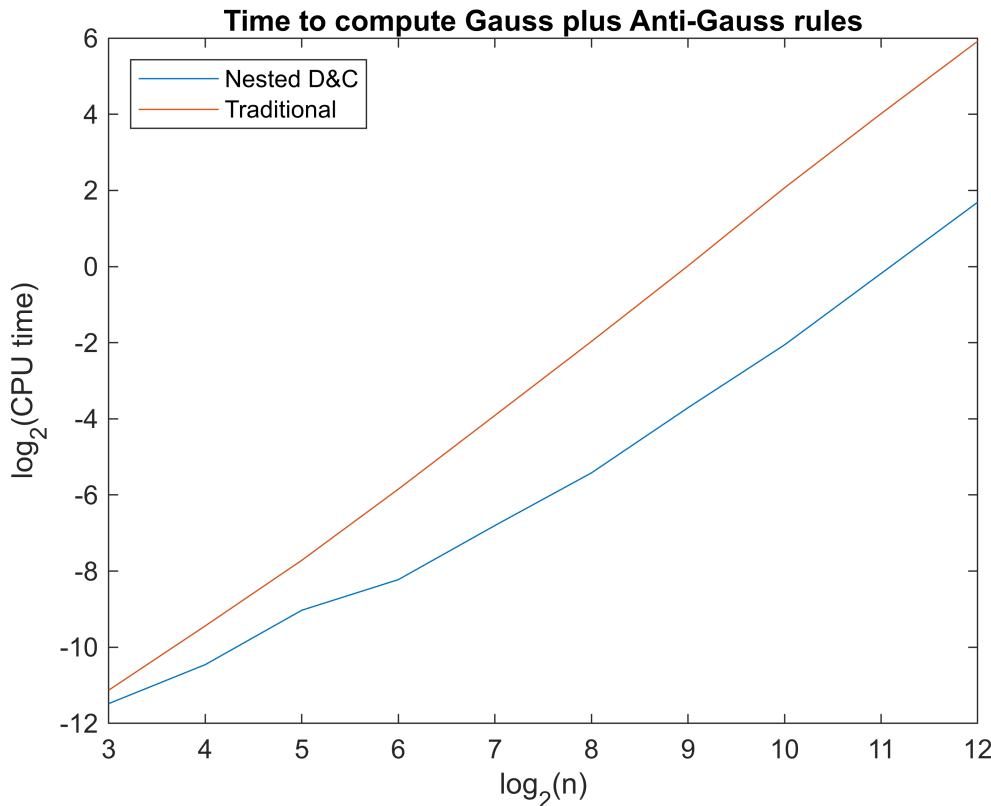
for k=logn
    n = 2^k;
    % Generate the Legendre recurrence coefficients.
    [alpha, beta] = JacobiRecurrence(n+1,0,0);
    % Create function handles to the two algorithms.
    nested = @( ) GaussPlusAntiGauss(alpha,beta); % Generate both rules using
    GaussPlusAntiGauss().
    trad = @( ) GaussPlusAntiGaussTraditional(alpha,beta); % Generate both rules
    using GaussPlusAntiGaussTraditional().

    nested_times(k) = timeit(nested);
    trad_times(k) = timeit(trad);
end
```

```

plot((1:MAX),log2(nested_times)); hold on;
plot((1:MAX),log2(trad_times));
xlabel('log2(n)');
ylabel('log2(CPU time)');
title('Time to compute Gauss plus Anti-Gauss rules');
legend('Nested D&C','Traditional','location','northwest');
hold off

```



## Comparing the Performance of the Nested D&C GaussPlusOptimalAveragedGauss code vs. the Traditional Approach

This section demonstrates the performance of the Nested D&C GaussPlusOptimalAveragedGauss code by using it to compute an  $n$ -point Gauss-Legendre and the  $2n+1$  point Optimal Averaged Gauss rule. It then compares this to a traditional approach where the  $n$ -point Gauss-Legendre rule and the  $2n+1$  point Optimal Averaged Gauss rule are both computed with independent applications of the TQR Golub-Welsch algorithm.

```

logn = 3:MAX;
nested_times = zeros(size(logn));
trad_times = zeros(size(logn));

for k=logn
    n = 2^k;
    % Generate the Legendre recurrence coefficients.
    [alpha, beta] = JacobiRecurrence(n+2,0,0);
    % Create function handles to the two algorithms.

```

```

    nested = @() GaussPlusOptimalAveragedGauss(alpha,beta); % Generate the rules
    using GaussPlusOptimalAveragedGauss().
    trad = @() GaussPlusOptimalAveragedGaussTraditional(alpha,beta); % Generate
    both rules using GaussPlusOptimalAveragedGaussTraditional().

    nested_times(k) = timeit(nested);
    trad_times(k) = timeit(trad);
end

plot((1:MAX),log2(nested_times)); hold on;
plot((1:MAX),log2(trad_times));
xlabel('log2(n)');
ylabel('log2(CPU time)');
title('Time to compute Gauss plus Optimal Averaged Gauss rules');
legend('Nested D&C','Traditional','location','northwest');
hold off

```

