```
% display code
type('rectrap.m');
type('compguassquad.m')

f = @statError;
a = 0;
b = 3;
n = 200;
err = 10^-5;
[S_t, error, evals] = rectrap(f, a, b, n, err);



fprintf(' Trapezoidal Rule\n');
fprintf('The value of the integral is %f\n', S_t);
fprintf('The number of function evaluations is %d\n', evals);
fprintf('The error is %f\n', error);


% quadrature
n = (evals + 1) / 2;
n = floor(n);

result = compguassquad(f, a, b, n);

fprintf(' Composite Gauss Quadrature\n');
fprintf('The value of the integral is %f\n', result);
fprintf('The number of function evaluations is %d\n', n);


% the real value is given by erf(3)
correct_val = erf(3);
fprintf('The correct value is %.10f\n', correct_val);
% the computed values
fprintf('The value of the integral using the trapezoidal rule is %.10f\n',
S_t);
fprintf('The value of the integral using the composite Gauss Quadrature is
%.10f\n', result);
% the error of both is given by
fprintf('The error of the trapezoidal rule is %.10f\n', abs(correct_val -
S_t));
fprintf('The error of the composite Gauss Quadrature is %.10f\n',
abs(correct_val - result));

% we can see the composite Gauss Quadrature  converges faster than the
trapezoidal rule
% because we needed much less function evaluations to get the same error


function [S_T, error, evals] = rectrap(f, a, b, n, er)
  % RECTRAP Recursive trapezoid function with caching
```

```
  h_T = b - a;
  S_T = 0.5 * h_T * (f(b) - f(a));


  cache = containers.Map({'key'}, {0});
  remove(cache, 'key');
  cache(num2str(a)) = f(a);
  cache(num2str(b)) = f(b);
  evaluations = 2;

  for i = 1:n
      h_i = h_T / 2;
      X = a + h_i : h_T : b - h_i;


      for j = 1:length(X)
          x_str = num2str(X(j));
          if ~isKey(cache, x_str)
              cache(x_str) = f(X(j));
              evaluations = evaluations + 1;
          end
          X(j) = cache(x_str);
      end

      S_i = 0.5 * S_T + h_i * sum(X);
      error = (S_i - S_T) / S_i;
      S_T = S_i;
      h_T = h_i;
      fprintf('S_%d = %f, error = %f\n', i, S_T, error);
      if abs(error) < er
          break
      end
  end

  evals = evaluations;

  end

% function [S_T, error, evals] = rectrap(f, a, b, n, er)
%  %RECTRAP Recursive trapezoid function
% %


% h_T = b - a;
% S_T = 0.5 * h_T * (f(b) - f(a)) ;



% evaluations = 2;

% for i = 1:n
%     h_i = h_T / 2 ;
%     X = a + h_i : h_T : b - h_i;
%     X = arrayfun(f, X);
```

```
%     evaluations = evaluations + length(X);
%     S_i = 0.5 * S_T + h_i * sum(X);
%     error = (S_i - S_T) / S_i;
%     S_T = S_i;
%     h_T = h_i;
%     fprintf('S_%d = %f, error = %f\n', i, S_T, error);
%     if abs(error) < er
%         break
%     end
% end

% evals = evaluations;

% end


function [result] = compguassquad(f, a, b, n)
%COMPGUASSQUAD
%   This computes the composite guassian quadrature using the two point rul
% it is composite, because we compute the sum of the gauss quadrature over
% sub intervals
% we divide out interval into n sub intervals
% [xi, xi+1], i = 0, 1, 2, ..., n-1
% we then apply the two point gauss quadrature to each sub interval

% recall the two point quadrature rule is as follow
% int [-1, 1] f(x) dx = f(-sqrt(3)/3) + f(sqrt(3)/3)

% we need to map the interval [a, b] to [-1, 1]
% we do this by the following transformation
% x = (b-a)/2 * t + (b+a)/2
% dx = (b-a)/2 * dt
% we then have
% beta = (b-a)/2
% alpha = (b+a)/2
% int [a, b] f(x) dx = beta * int [-1, 1] f(beta * t + alpha) dt
% this is roughly equal to
% beta *  sum [0,n] A_i * f(beta * t_i + alpha)

% for use and our two point quadrature rule this means we have

% beta = (b-a)/2
% alpha = (b+a)/2
% int [a, b] f(x) dx = beta * (f(beta * (-sqrt(3)/3) + alpha) + f(beta *
(sqrt(3)/3) + alpha))

% we then need to compute the sum of the two point quadrature over the
% subintervals

% so we will our formula is
% h = (b-a)/n
% beta_i = (x_i+1 - x_i)/2 = h/2
% alpha_i = (x_i+1 + x_i)/2 = a + i*h + h/2
```

```
% sum [0,n-1] int [xi, xi+1] f(x) dx = sum [0,n-1] beta_i * (f(beta_i *
(-sqrt(3)/3) + alpha_i) + f(beta_i * (sqrt(3)/3) + alpha_i))

result = 0;

h = (b-a)/n;
beta = h/2;
beta_sqrt = beta * sqrt(3)/3;


for i = 0:n-1
    alpha = a + i*h + h/2;
    result = result + (f(beta_sqrt + alpha) + f(-beta_sqrt + alpha));
end


result = beta * result;
end

S_1 = -0.667785, error = -1.534290
S_2 = 0.153663, error = 5.345779
S_3 = 0.576827, error = 0.733607
S_4 = 0.788404, error = 0.268362
S_5 = 0.894192, error = 0.118305
S_6 = 0.947085, error = 0.055848
S_7 = 0.973531, error = 0.027166
S_8 = 0.986755, error = 0.013401
S_9 = 0.993366, error = 0.006656
S_10 = 0.996672, error = 0.003317
S_11 = 0.998325, error = 0.001656
S_12 = 0.999151, error = 0.000827
S_13 = 0.999565, error = 0.000413
S_14 = 0.999771, error = 0.000207
S_15 = 0.999875, error = 0.000103
S_16 = 0.999926, error = 0.000052
S_17 = 0.999952, error = 0.000026
S_18 = 0.999965, error = 0.000013
S_19 = 0.999971, error = 0.000006
 Trapezoidal Rule
The value of the integral is 0.999971
The number of function evaluations is 127478
The error is 0.000006
 Composite Gauss Quadrature
The value of the integral is 0.999978
The number of function evaluations is 63739
The correct value is 0.9999779095
The value of the integral using the trapezoidal rule is 0.9999714563
The value of the integral using the composite Gauss Quadrature is 0.9999779095
The error of the trapezoidal rule is 0.0000064532
The error of the composite Gauss Quadrature is 0.0000000000
```

Published with MATLAB® R2023b