EPFL

Numerical Analysis and Computational Mathematics

Fall Semester 2025 – CSE Section

Prof. Laura Grigori

Assistant: Israa Fakih

Session 7 – October 29, 2025

Solutions – Numerical differentiation and integration

Solution I (MATLAB)

a) We implement the MATLAB functions as follows:

```
function [ dfh ] = forward_finite_difference( fun, xnodes, h )
% FORWARD_FINITE_DIFFERENCE approximate the first derivative of a function
% in the nodes by using the forward finite difference scheme
%
%  [ dfh ] = forward_finite_difference( fun, xnodes, h )
%  Inputs: fun = function handle,
%          xnodes = vector of nodes' coordinates
%          h = coordinates increment; positive and scalar value.
%  Output: dfh = approximate values of the first derivatives of fun in the
%          nodes.
%

dfh = ( fun( xnodes + h ) - fun( xnodes ) ) / h;

return
```

```
function [ dfh ] = backward_finite_difference( fun, xnodes, h )
% BACKWARD_FINITE_DIFFERENCE approximate the first derivative of a function
% in the nodes by using the backward finite difference scheme
%
%  [ dfh ] = backward_finite_difference( fun, xnodes, h )
%  Inputs: fun = function handle,
%          xnodes = vector of nodes' coordinates
%          h = coordinates increment; positive and scalar value.
%  Output: dfh = approximate values of the first derivatives of fun in the
%          nodes.
%

dfh = ( fun( xnodes ) - fun( xnodes - h ) ) / h;
```

```
return
```

```
function [ dfh ] = centered_finite_difference( fun, xnodes, h )
% CENTERED_FINITE_DIFFERENCE approximate the first derivative of a function
% in the nodes by using the centered finite difference scheme
%
%  [ dfh ] = centered_finite_difference( fun, xnodes, h )
%  Inputs: fun = function handle,
%          xnodes = vector of nodes' coordinates
%          h = coordinates increment; positive and scalar value.
%  Output: dfh = approximate values of the first derivatives of fun in the
%          nodes.
%

dfh = ( fun( xnodes + h ) - fun( xnodes - h ) ) / ( 2 * h );

return
```

b) We use the following MATLAB commands:

```
f = @( x ) x .* log( x ) - sin( x ).^2 ;
df = @( x ) 1 + log( x ) - 2 * sin( x ) .* cos( x );
xnode = 1.9;
df_exact_node = df( xnode )
%  df_exact_node =
%      2.2537
h = 1 / 16;
[ dfh_f_node ] = forward_finite_difference( f, xnode, h )
%  dfh_f_node =
%      2.3178
[ dfh_b_node ] = backward_finite_difference( f, xnode, h )
%  dfh_b_node =
%      2.1861
[ dfh_c_node ] = centered_finite_difference( f, xnode, h )
%  dfh_c_node =
%     2.2519
```

c) We use the following MATLAB commands to obtain the results reported in Figure 1:

```
h_vect = 2.^[ -2 : -1 : -7 ];
err_f = [ ]; err_b = [ ];  err_c = [ ];
for h = h_vect
    [ dfh_f_node ] = forward_finite_difference( f, xnode, h );
    err_f = [ err_f, abs( df_exact_node - dfh_f_node ) ];
    [ dfh_b_node ] = backward_finite_difference( f, xnode, h );
    err_b = [ err_b, abs( df_exact_node - dfh_b_node ) ];
    [ dfh_c_node ] = centered_finite_difference( f, xnode, h );
    err_c = [ err_c, abs( df_exact_node - dfh_c_node ) ];
end
viz_factor = 1.5; % factor to improve the visualization of the line h
loglog( h_vect, err_f, '-ob', h_vect, err_b, '-xr', h_vect, err_c, '-sm', ...
     h_vect, viz_factor * h_vect, '--k', h_vect, h_vect.^2, '-.k' );
grid on;
xlabel('h'); ylabel('err');
```

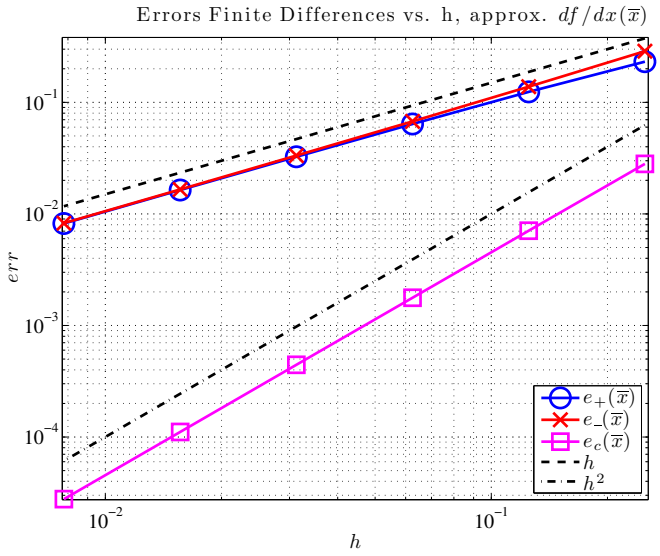


Figure 1: Errors $(e_+f)(\overline{x})$, $(e_-f)(\overline{x})$, and $(e_cf)(\overline{x})$ vs. h for the forward, backward, and centered finite differences, respectively. Comparison with h and h^2.

```
legend( 'e_{+}', 'e_{-}', 'e_{c}', 'h', 'h^2');
title('Error Finite Differences vs. h, approx df/dx')
```

The errors (e_+f), (e_-f), and (e_cf) corresponding to the forward, backward, and centered finite differences schemes are, for a sufficiently regular function $f(x)$:

$$f'(\overline{x}) - (\delta_+ f)(\overline{x}) = -\frac{h}{2} f''(\xi), \qquad \text{for some } \xi \in [\overline{x}, \overline{x}+h],$$

$$f'(\overline{x}) - (\delta_- f)(\overline{x}) = \frac{h}{2} f''(\eta), \qquad \text{for some } \eta \in [\overline{x}-h, \overline{x}],$$

$$f'(\overline{x}) - (\delta_c f)(\overline{x}) = -\frac{h^2}{12} (f'''(\zeta^+) + f'''(\zeta^-)), \qquad \text{for some } \zeta^+ \in [\overline{x}, \overline{x}+h],\ \zeta^- \in [\overline{x}-h, \overline{x}].$$

Since, in this case, $f(x) \in C^\infty([a,b])$ (including a neighborhood of $\overline{x}$), we expect the orders of convergence for the errors to be 1, 1, and 2 for the forward, backward, and centered finite differences schemes, respectively. This is confirmed in Figure 1.

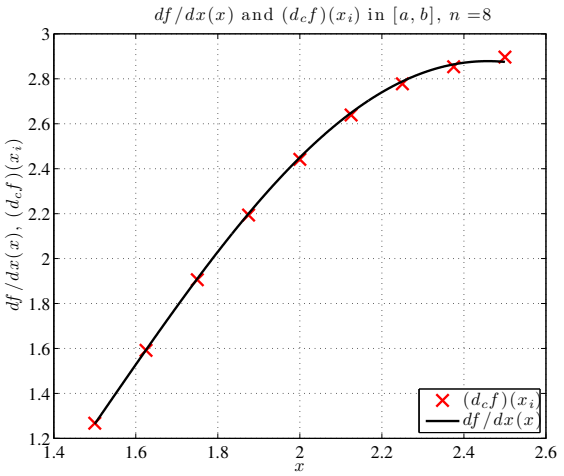
Moreover, we can calculate numerically the convergence orders by computing the errors for two values of h. Let $(e_\bullet f)_h$ be the magnitude of the error corresponding to a generic finite difference scheme $\bullet$ for a given h and assume that it can be expressed as $(e_\bullet f)_h = C\,h^\alpha$ with $\alpha > 0$ and a positive constant C independent of h and α. Then, we can estimate α as $\alpha = \log_\beta\left((e_\bullet f)_{h_2}/(e_\bullet f)_{h_1}\right) / \log_\beta(h_2/h_1)$, for any $\beta > 1$ and $h_1 \neq h_2$ "sufficiently small". By using the following MATLAB commands, we confirm that the order of convergence of the errors associated to the forward, backward, and centered finite difference schemes agree with the expected ones:

```
conv_order_f = log( err_f( end - 1 ) / err_f( end ) ) / ...
                        log( h_vect( end - 1 ) / h_vect( end ) )
%  conv_order_f =
```

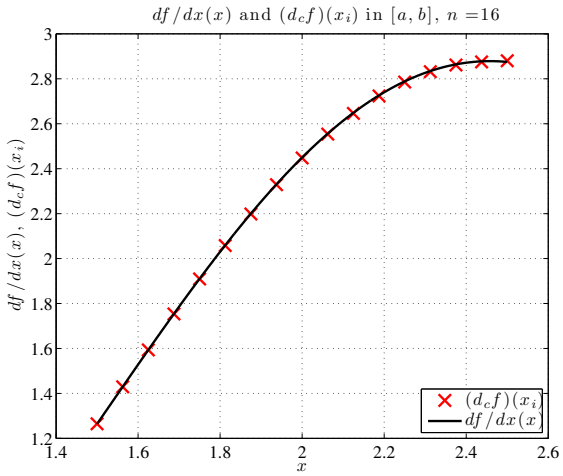
```
%       0.9951
conv_order_b = log( err_b( end - 1 ) / err_b( end ) ) / ...
                           log( h_vect( end - 1 ) / h_vect( end ) )
%  conv_order_b =
%       1.0048
conv_order_c = log( err_c( end - 1 ) / err_c( end ) ) / ...
                           log( h_vect( end - 1 ) / h_vect( end ) )
%   conv_order_c =
%       2.0000
```

d) Given $n = 8$, we use the following MATLAB commands to obtain the result reported in Figure 2 (left).

```
a = 3 / 2;  b = 5 / 2;
n = 8; % n is the number of intervals, n + 1 nodes
h = ( b - a ) / n;
x_nodes = linspace( a, b, n + 1 );
x_nodes_internal = x_nodes( 2 : end - 1 );
[ dfh_c_nodes_internal ] = centered_finite_difference( f, x_nodes_internal, h );
dfh_c_node_a = ( - 3 * f( a ) + 4 * f( a + h ) - f( a + 2 * h ) ) / ( 2 * h );
dfh_c_node_b = ( 3 * f( b ) - 4 * f( b - h ) + f( b - 2 * h ) ) / ( 2 * h );
dfh_c_nodes = [ dfh_c_node_a, dfh_c_nodes_internal, dfh_c_node_b ];
x_values = linspace( a, b , 100 * ( n + 1 ) ); % for visualization of exact values
df_exact_values = df( x_values );
figure; plot( x_nodes, dfh_c_nodes, 'xr', x_values, df_exact_values, '-k' );
legend( '(d_c f)(x_i)', 'df/dx(x)');
```



$n = 8$



$n = 16$

Figure 2: Derivative $f'(x)$ for $x \in [3/2, 5/2]$ and its approximation in the nodes $\overline{x}_j$, $j = 0, \ldots, n$ by using the centered finite differences.

We observe that already for this choice of n (with $h = (b-a)/n$), the first derivative of the function $f(x)$ at the nodes $\overline{x}_j$ is approximated fairly well. We repeat the previous MATLAB command to obtain the result reported in Figure 2 (right).

Solution II (MATLAB)

a) We implement the MATLAB functions as follows:

```
function [ Ih ] = midpoint_composite_quadrature( fun, a, b, M )
% MIDPOINT_COMPOSITE_QUADRATURE approximate the integral of a function in
% the interval [a,b] by means of the composite midpoint quadrature formula
%  [ Ih ] = midpoint_composite_quadrature( fun, a, b, M )
%  Inputs: fun = function handle,
%          a,b = extrema of the interval [a,b]
%          M = number of subintervals of [a,b] of the same size, M≥1
%              (the case M=1 corresponds to the simple formula)
%  Output: Ih = approximate value of the integral
%

H = ( b - a ) / M;

x_k = linspace( a, b, M + 1 );  % M+1 nodes
x_bar_k = ( x_k( 1 : end - 1 ) + x_k( 2 : end ) ) / 2; % M coordinates

f_x_bar_k = fun( x_bar_k );  % M values

Ih = H * sum( f_x_bar_k );

return
```

```
function [ Ih ] = trapezoidal_composite_quadrature( fun, a, b, M )
% TRAPEZOIDAL_COMPOSITE_QUADRATURE approximate the integral of a function in
% the interval [a,b] by means of the composite trapezoidal quadrature formula
%  [ Ih ] = trapezoidal_composite_quadrature( fun, a, b, M )
%  Inputs: fun = function handle,
%          a,b = extrema of the interval [a,b]
%          M = number of subintervals of [a,b] of the same size, M≥1
%              (the case M=1 corresponds to the simple formula)
%  Output: Ih = approximate value of the integral
%

H = ( b - a ) / M;

x_k = linspace( a, b, M + 1 ); % M+1 nodes

f_x_km1 = fun( x_k( 1 : end - 1 ) ); % M values, in x_{k-1}
f_x_k = fun( x_k( 2 : end ) ); % M values, in x_{k}

Ih = H / 2 * sum( f_x_km1 + f_x_k );

return
```

```
function [ Ih ] = simpson_composite_quadrature( fun, a, b, M )
% SIMPSON_COMPOSITE_QUADRATURE approximate the integral of a function in
% the interval [a,b] by means of the composite Simpson quadrature formula
%  [ Ih ] = simpson_composite_quadrature( fun, a, b, M )
%  Inputs: fun = function handle,
```

```
%          a,b = extrema of the interval [a,b]
%          M = number of subintervals of [a,b] of the same size, M≥1
%              (the case M=1 corresponds to the simple formula)
%  Output: Ih = approximate value of the integral
%

H = ( b - a ) / M;

x_k = linspace( a, b, M + 1 ); % M+1 nodes
x_bar_k = ( x_k( 1 : end - 1 ) + x_k( 2 : end ) ) / 2; % M coordinates

f_x_km1 = fun( x_k( 1 : end - 1 ) ); % M values, in x_{k-1}
f_x_k = fun( x_k( 2 : end ) ); % M values, in x_{k}
f_x_bar_k = fun( x_bar_k ); % M values, in x_{k_bar}

Ih = H / 6 * sum( f_x_km1 + 4 * f_x_bar_k + f_x_k );

return
```

b) We use the following MATLAB commands, with $M = 1$:

```
f = @(x) sin( 7/2 * x ) + exp( x ) - 1;   a = 0;   b = 1;
I_ex = exp( 1 ) - 2 + 2 / 7 * ( 1 - cos( 7 / 2 ) )
%   I_ex =
%        1.2716
M = 1;
Imp = midpoint_composite_quadrature( f, a, b, M )
%    Imp =
%       1.6327
It = trapezoidal_composite_quadrature( f, a, b, M )
%    It =
%        0.6837
Is = simpson_composite_quadrature( f, a, b, M )
%    Is =
%        1.3164
```

We observe that, in this case, the Simpson formula returns an approximated value of the integral closer to the exact value $I(f)$ than the midpoint and trapezoidal formulas.

c) We use the following MATLAB commands, with $M = 10$:

```
M = 10;
Imp_c = midpoint_composite_quadrature( f, a, b, M )
%    Imp_c =
%        1.2737
It_c = trapezoidal_composite_quadrature( f, a, b, M )
%    It_c =
%        1.2673
Is_c = simpson_composite_quadrature( f, a, b, M )
%    Is_c =
%        1.2716
```

d) We use the following MATLAB commands to obtain the results reported in Figure 3:

```
M_vect = 2.^[2:1:7];
H_vect = ( b - a ) ./ M_vect;
Err_mp_c = [ ];  Err_t_c = [ ];   Err_s_c = [ ];
for M = M_vect
    Imp_c = midpoint_composite_quadrature( f, a, b, M );
    Err_mp_c = [ Err_mp_c, abs( Imp_c - I_ex ) ];
    It_c = trapezoidal_composite_quadrature( f, a, b, M );
    Err_t_c = [ Err_t_c, abs( It_c - I_ex ) ];
    Is_c = simpson_composite_quadrature( f, a, b, M );
    Err_s_c = [ Err_s_c, abs( Is_c - I_ex ) ];
end
% Plot of the errors and the reference curves H^2 and H^4
loglog( H_vect, Err_mp_c, '-ob', H_vect, Err_t_c, '-xr', H_vect, Err_s_c, '-sm',...
            H_vect, H_vect.^2, '--k', H_vect, H_vect.^4, '-.k' );
legend( 'E_{mp}(f)', 'E_{t}(f)', 'E_{s}(f)', 'H^2', 'H^4');
```

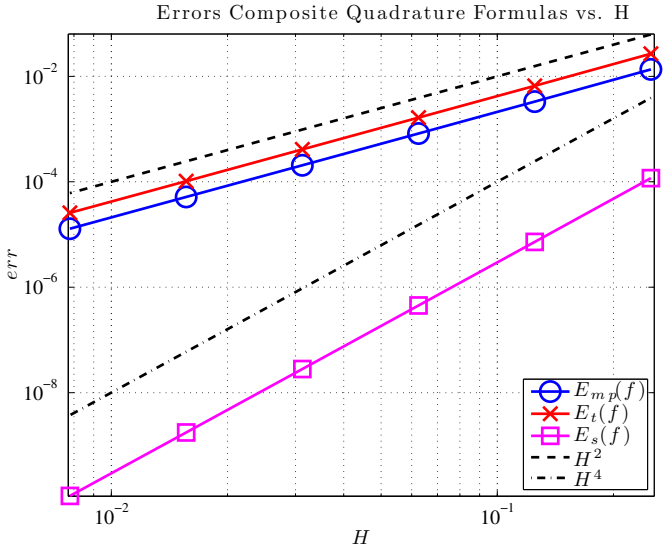


Figure 3: Errors $E^c_{mp}(f)$, $E^c_t(f)$, and $E^c_s(f)$ vs. H for the composite midpoint, trapezoidal, and Simpson quadrature formulas, respectively. Comparison with the lines (H, H^2) and (H, H^4).

We recall that, for a sufficiently regular function $f(x)$, we have

$$e^c_{mp}(f) = I(f) - I^c_{mp}(f) = \frac{b-a}{24} H^2 f''(\xi), \qquad \text{for some } \xi \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e^c_t(f) = I(f) - I^c_t(f) = -\frac{b-a}{12} H^2 f''(\eta), \qquad \text{for some } \eta \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e^c_s(f) = I(f) - I^c_s(f) = -\frac{b-a}{16 \cdot 180} H^4 f^{(4)}(\zeta), \qquad \text{for some } \zeta \in [a,b], \qquad \text{if } f \in C^4([a,b]).$$

Since, in this case, $f(x) \in C^\infty([a,b])$, we expect the orders of accuracy (convergence orders of the errors) to be equal to 2, 2, and 4 for the composite midpoint, trapezoidal, and Simpson quadrature formulas, respectively. This is confirmed in Figure 3.

We use the following MATLAB commands to compute numerically the convergence orders:

```
conv_order_mp_c = log( Err_mp_c( end ) / Err_mp_c( end - 1 ) ) / ...
                        log( H_vect( end ) / H_vect( end - 1 ) )
%   conv_order_mp_c =
%       2.0001
conv_order_t_c = log( Err_t_c( end ) / Err_t_c( end - 1 ) ) / ...
                        log( H_vect( end ) / H_vect( end - 1 ) )
%   conv_order_t_c =
%       2.0001
conv_order_s_c = log( Err_s_c( end ) / Err_s_c( end - 1 ) ) / ...
                        log( H_vect( end ) / H_vect( end - 1 ) )
%   conv_order_s_c =
%       4.0001
```

e) The function $f(x) = x^d$ is a polynomial of degree d. The simple midpoint, trapezoidal, and Simpson quadrature formulas have degree of exactness equal to 1, 1, and 3, respectively. We deduce the results from the corresponding errors:

$$e_{mp}(f) = I(f) - I_{mp}(f) = \frac{(b-a)^3}{24} f''(\xi), \qquad \text{for some } \xi \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e_t(f) = I(f) - I_t(f) = -\frac{(b-a)^3}{12} f''(\eta), \qquad \text{for some } \eta \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e_s(f) = I(f) - I_s(f) = -\frac{(b-a)^5}{16 \cdot 180} f^{(4)}(\zeta), \qquad \text{for some } \zeta \in [a,b], \qquad \text{if } f \in C^4([a,b]).$$

Indeed, $f''(x) = 0$ for any polynomial of degree $d \leq 1$ and $f^{(4)}(x) = 0$ for any polynomial of degree $d \leq 3$.

We consider the following MATLAB commands to verify the above results:

```
f = @(x,d) x.^d; a=0;  b=1;
Table = [ ];
for d = [ 0 : 4 ]
    I_ex = 1 / ( d + 1 );  % exact integral
    Imp = midpoint_composite_quadrature( @(x)f(x,d), a, b, 1 );
    It = trapezoidal_composite_quadrature( @(x)f(x,d), a, b, 1 );
    Is = simpson_composite_quadrature( @(x)f(x,d), a, b, 1 );
    err_mp = I_ex - Imp;     err_t = I_ex - It;        err_s = I_ex - Is;
    Table = [ Table; [ d, err_mp, err_t, err_s ] ];
end
disp('         d, err_{mp},    err_t,    err_s'); disp( Table );

%            d, err_{mp},    err_t,    err_s
%            0         0         0         0
%       1.0000         0         0         0
%       2.0000    0.0833   -0.1667         0
%       3.0000    0.1250   -0.2500         0
%       4.0000    0.1375   -0.3000   -0.0083
```

From the variable `Table`, we verify that the errors for all the formulas are zero for polynomials $f(x) = x^d$ of degree $d \leq 1$, while the errors associated to the simple Simpson quadrature formula are zero for all degrees $d \leq 3$. The results are in agreement with the degrees of exactness expected for these quadrature formulas.

Solution III (Theoretical)

a) We have:

$$e_{mp}(f) = \frac{(b-a)^3}{24} f''(\xi), \qquad \text{for some } \xi \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e_t(f) = -\frac{(b-a)^3}{12} f''(\eta), \qquad \text{for some } \eta \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e_s(f) = -\frac{(b-a)^5}{16 \cdot 180} f^{(4)}(\zeta), \qquad \text{for some } \zeta \in [a,b], \qquad \text{if } f \in C^2([a,b]).$$

We set $f(x) = f_1(x)$, $a = 0$, and $b = 1$, and we obtain $f_1''(x) = 8$ and $f^{(4)}(x) = 0$. Accordingly, $e_{mp}(f_1) = \frac{1}{3}$, $e_t(f_1) = -\frac{2}{3}$, and $e_s(f_1) = 0$ since the Simpson quadrature formula has degree of exactness equal to 3.

b) For M uniform sub-intervals of size H, we have:

$$e^c_{mp}(f) = \frac{b-a}{24} H^2 f''(\xi), \qquad \text{for some } \xi \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e^c_t(f) = -\frac{b-a}{12} H^2 f''(\eta), \qquad \text{for some } \eta \in [a,b], \qquad \text{if } f \in C^2([a,b]),$$

$$e^c_s(f) = -\frac{b-a}{16 \cdot 180} H^4 f^{(4)}(\zeta), \qquad \text{for some } \zeta \in [a,b], \qquad \text{if } f \in C^4([a,b]).$$

We deduce that:

$$|e^c_{mp}(f)| \le \widetilde{e}^c_{mp}(f) := \frac{b-a}{24} H^2 \max_{x\in[a,b]} |f''(x)|,$$

$$|e^c_t(f)| \le \widetilde{e}^c_t(f) := \frac{b-a}{12} H^2 \max_{x\in[a,b]} |f''(x)|,$$

$$|e^c_s(f)| \le \widetilde{e}^c_s(f) := \frac{b-a}{16 \cdot 180} H^4 \max_{x\in[a,b]} \left|f^{(4)}(x)\right|.$$

In order to find the minimum number of subintervals ensuring errors $< tol$, we solve the inequalities $\widetilde{e}^c_{mp}(f) < tol$, $\widetilde{e}^c_t(f) < tol$, and $\widetilde{e}^c_s(f) < tol$. The resulting values of M_{min} are

$$M_{min,mp} > \left(\frac{(b-a)^3}{24\,tol} \max_{x\in[a,b]} |f''(x)|\right)^{1/2},$$

$$M_{min,t} > \left(\frac{(b-a)^3}{12\,tol} \max_{x\in[a,b]} |f''(x)|\right)^{1/2},$$

$$M_{min,s} > \left(\frac{(b-a)^5}{16 \cdot 180\,tol} \max_{x\in[a,b]} |f^{(4)}(x)|\right)^{1/4}.$$

We set $f(x) = f_1(x)$, $a = 0$, and $b = 1$. We obtain that $f_1''(x) = 8$ and $f_1^{(4)}(x) = 0$. We deduce that for $f_1(x)$, $M_{min,mp} = 183$ for the composite midpoint formula and $M_{min,t} = 259$ for composite trapezoidal formula. Since the composite Simpson quadrature formula has degree of exactness 3 and $f_1(x)$ is a polynomial of degree 2, the formula returns the exact integral

$I(f_1)$ for any choice of $M \geq 1$, so that we may set $M_{min,s} = 1$, i.e. the simple Simpson formula.

We repeat all this for $f = f_2$. We have $f_2^{(k)}(x) = e^x$ for any $k \geq 2$. By setting $a = 0$ and $b = 1$, we obtain that $\max_{x\in[a,b]} |f''(x)| = \max_{x\in[a,b]} \left|f^{(4)}(x)\right| = e$. Hence, by using the notation previously introduced, we obtain that $M_{min,mp} = 107$, $M_{min,t} = 151$, and $M_{min,s} = 4$.

In both cases, due to the larger order of accuracy of the Simpson formula, the constants involved, and the fact that $\max_{x\in[a,b]} |f''(x)| = \max_{x\in[a,b]} |f^{(4)}(x)|$, the minimum number of subdivisions M_{min} required by the Simpson formula is considerably lower than with the other formulas.

Solution IV (Theoretical)

Since $I_t(f) = I(\Pi_1 f)$, we have $e_t(f) = I(f - \Pi_1 f) = \int_a^b (f(x) - \Pi_1 f(x))dx$. We recall that the truncation error $E_1 f(x)$ for linear polynomial interpolation can be expressed as

$$E_1 f(x) = \frac{1}{2} f''(\eta(x))\omega_1(x)$$

for some $\eta(x) \in [a, b]$ and $\omega_1(x) = (x - a)(x - b)$. Since $\omega_1(x)$ has constant sign, by using the mean value theorem for integrals, we have

$$e_t(f) = \frac{1}{2}\int_a^b f''(\eta(x))\omega_1(x)dx = \frac{1}{2} f''(\xi)\int_a^b \omega_1(x)dx,$$

for some $\xi \in [a, b]$. Then, since $\int_a^b \omega_1(x)dx = -\frac{(b-a)^3}{6}$, the result follows.