ADVANCED NUMERICAL METHODS

BACHELOR DEGREE IN INDUSTRIAL TECHNOLOGY ENGINEERING

JUNE 30, 2016

FULL ANSWER

Disclaimer: in a real exam situation the student is not supposed to give explanations nearly as complete as the ones below, whose main purpose is a teaching one, not an assessment-related one.

EXERCISE 1 **(13 points)**

A)

For a polynomial of degree ≤ 3 we need to choose 4 of the 11 possible nodes. To decide which ones we expect to result in the smallest error at $x = 2.55$, we look at the truncation error term:

$$e(x) = f[x_0, x_1, ..., x_n, x]\Pi(x) = \frac{f^{n+1)}(\xi)}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$$

for some ξ between the nodes and x.

Unless there is some coincidence with ξ (whose value is most often left unknown) such that $|f^{n+1)}(\xi)|$ turns out to be very small, we normally expect $|e(x)|$ to be small when the nodes make $|\Pi(x)|$ small, i.e. when the (product of the) distances $|x–x_0|$, $|x–x_1|$, …, $|x–x_n|$ are small. So a priori we expect to get the smallest $|e(x)|$ with the closest nodes to x. This is a general rule of thumb: the closer the nodes are to x, the smaller $|e(x)|$ tends to be.

In our case, a priori we expect the truncation error at $x = 2.55$ to be smallest when we choose the four eligible nodes closest to x, which are 2.4, 2.5, 2.6 and 2.7. Let us call them x_0, x_1, x_2, x_3, respectively. These nodes are evenly spaced, so to calculate the corresponding interpolation polynomial we can use finite differences. The table of finite differences with x_0, x_1, x_2, x_3 is, with Octave's help:

```
format long g
f = @log; xi = 2.4:0.1:2.7
     xi =             2.4                2.5                2.6                2.7
fi = f(xi)
     fi = 0.8754687373539 0.916290731874155 0.955511445027436 0.993251773010283
fi = anm_signif(fi, 5)                      % 5 significant digits
     fi = 0.87547         0.91629           0.95551           0.99325
[Dif0, table] = anm_tablefd(xi, fi);
prettyprintt(table, {'i', 'xi', 'fi', 'Dfi', 'D2fi', 'D3fi'})
     ans =
     i    xi        fi        Dfi        D2fi       D3fi
     -   ---   -------   -------   --------   -------
     0   2.4   0.87547       NaN        NaN        NaN
     1   2.5   0.91629   0.04082        NaN        NaN
     2   2.6   0.95551   0.03922    -0.0016        NaN
     3   2.7   0.99325   0.03774   -0.00148    0.00012
```

To write the Newton interpolation polynomial we will use the change of variable:

$$x = x_0 + h\,t = 2.4 + 0.1\,t \quad \Rightarrow \quad t(x) = (x–x_0)/h = (x–2.4)/0.1 = t = 10\,x – 24$$

The polynomial is $$\underline{p_a(x) = q(t(x)) = q(10x-24)}$$

where $$q(t) = 0.87547 + 0.04082\,t/1! – 0.00160\,t\,(t–1)/2! + 0.00012\,t\,(t–1)(t–2)/3! =$$

$$\underline{q(t) = 0.87547 + 0.04082\,t – 0.00080\,t\,(t–1) + 0.00002\,t\,(t–1)(t–2)}$$

N.B. As said above, this is a priori. To be totally sure that the smallest error at $x = 2.55$ is obtained with those four nodes, one thing we can do is ask Octave to calculate it for us using brute force. The following code calculates the error for all possible combinations of the 11 possible nodes choose 4 (there are 330 such combinations), and confirms our prior expectation:

```
clear, format short g
xiall = 2:0.1:3               % all possible nodes
     xiall =   2   2.1   2.2   2.3   2.4   2.5   2.6   2.7   2.8   2.9   3
xi = xiall(5:8)               % nodes expected a priori to be the best
     xi =       2.4          2.5          2.6          2.7
x = 2.55; f = @log;
errsmall = f(x) - anm_newton(x, xi, f(xi))               % error to beat
     errsmall =  -3.3344e-007
iii = combnk(1:11, 4);        % all combinations of elements 1:11 choose 4
nc = length(iii)              % number of such combinations
     nc =        330
errs = zeros(1, nc);          % a place for the error with each node combination
for j = 1:nc, xi = xiall(iii(j,:)); errs(j) = f(x) - anm_newton(x,xi,f(xi)); end
find(abs(errs) <= abs(errsmall))  % see how many times we got errsmall or better
     ans =       35
xiall(iii(ans,:))                  % only once! The nodes, as expected, are:
     ans =      2.4          2.5          2.6          2.7
```

B)

The error made is the exact value minus the approximate one. The "exact" value (within the 5-significant digit arithmetic used) is:

$$f(2.55) = \log(2.55) = 0.93609335917\ldots \approx 0.93609$$

The approximate value is $p_a(2.55)$. For $x = 2.55$, $t = 10x–24 = 25.5–24 = 1.5$. Let us evaluate $p_a(2.55) = q(1.5)$ with the Horner-like algorithm (since we are told to do this optimally):

$$q(t) = 0.87547 + t\,\{0.04082 + (t–1)\,[–0.00080 + (t–2)\,0.00002]\}$$

$$q(1.5) = \underbrace{0.87547 + \underbrace{1.5\times\{\underbrace{0.04082 + \underbrace{0.5\times[\underbrace{-0.00080 + \underbrace{(-0.5)\times 0.00002}_{-0.00001}}_{-0.00081}]}_{-0.000405}}_{0.040415}\}}_{0.0606225\approx 0.060623}}_{0.936093\approx 0.93609} \approx 0.93609$$

Hence: $$e_a(2.55) = \log(2.55) - p_a(2.55) \approx 0.93609 - 0.93609 = \underline{0.0}$$

But, of course, the error is not really zero! We get this value because of the limited precision of the arithmetic used. The digits of the truncation error (i.e., the error with exact arithmetic) must be to the right of the 5 digits of precision we are allowed to use. (With double-precision arithmetic throughout, the error can actually was found to be about -3.3344×10^{-7} above, which is about two decimal places to the right of the digits we are allowed to consider.)

C)

The truncation error (i.e. the error with exact arithmetic) can actually be estimated using 5-significant-digit arithmetic! The estimation of the error we will use is:

$$e_a(2.55) \approx p_4(2.55) - p_a(2.55)$$

where p_4 is the polynomial by the four nodes initially chosen plus the new node $x_4 = 2.8$. This is because $p_4(x)$ is supposed to be a better estimation of $f(x)$ than $p_3(x)$.

To calculate p_4 we can just add one row to the table of divided differences above:

```
xi = 2.4:0.1:2.8, f = @log;
     xi =        2.4        2.5        2.6        2.7        2.8
fi = anm_signif(f(xi), 5)
     fi =    0.87547    0.91629    0.95551    0.99325    1.0296
[Dif0, table] = anm_tablefd(xi, fi);
prettyprintt(table, {'i', 'xi', 'fi', 'Dfi', 'D2fi', 'D3fi', 'D4fi'})
     ans =
     i    xi        fi        Dfi        D2fi       D3fi     D4fi
     -    ---    -------    -------    --------    -------    -----
     0    2.4    0.87547       NaN         NaN        NaN      NaN
     1    2.5    0.91629    0.04082         NaN        NaN      NaN
     2    2.6    0.95551    0.03922     -0.0016        NaN      NaN
     3    2.7    0.99325    0.03774    -0.00148    0.00012      NaN
     4    2.8     1.0296    0.03635    -0.00139       9e-5    -3e-5
```

Only the last row is new here.

The term to be added to $p_a(x) = p_3(x)$ in order to obtain $p_4(x)$ is called $h_4(x)$ in our theory. From the last element in the last row, that is:

$$h_4(x(t)) = p_4(x(t)) - p_3(x(t)) = -0.00003\, t\,(t-1)(t-2)(t-3)\,/\,4!$$

Its value at $t = t(2.55) = 1.5$ is the estimation requested:

$$e_a(2.55) \approx -0.00003\times 1.5\times 0.5\times(-0.5)(-1.5)\,/\,4! = \underline{-7.0313\times10^{-7}}$$

A priori I would say that this estimation of the error should be fairly good, because the new node is not so far from $x = 2.55$ that it cannot provide useful information on x; because the derivatives of $f(x) = \log(x)$ are all well-behaved in $[2.4, 2.9]$, in that negative powers of x do not undergo wild or weird variations in this interval; and because cancellation errors are not apparent even with the limited-precision arithmetic used.

(As found above, the truncation error is actually about -3.33×10^{-7}, so the estimation is indeed not bad at all considering that we got it using 5-significant-digit arithmetic. In fact, with double-precision arithmetic, the estimation of the error as $p_4(x)-p_a(x)$ would have been about -3.09×10^{-7}, which is even closer to -3.33×10^{-7}.)

D)

In our case: $$e(x) = \frac{f^{4)}(\xi)}{4!}(x-2.4)(x-2.5)(x-2.6)(x-2.7)$$

with $$f(x) = \log(x)\,;\quad f'(x) = x^{-1}\,;\quad f''(x) = -x^{-2}\,;\quad f^{3)}(x) = 2x^{-3}\,;\quad f^{4)}(x) = -6/x^4$$

The function $f^{4)}$ is monotonic in $[2, 3]$, so it will be bounded by its values at the endpoints:

$$f^{4)}(2) = -0.375\,;\quad f^{4)}(3) = -0.074074 \quad\Rightarrow\quad |f^{4)}(\xi)| \le 0.375$$

because ξ is between the nodes and x, so in our case $\xi \in [2, 3]$.

As for $|\Pi(x)|$, which is the product of the distances $|x - x_i|$ from x to the four nodes, its maximum in $[2, 3]$ is necessarily at $x = 2$, because that is the point of $[2, 3]$ farthest from all four nodes as a whole:

$$\Pi(x) \le \Pi(2) = (2-2.4)(2-2.5)(2-2.6)(2-2.7) = 0.084$$

Substituting: $$|e(x)| = \left|\frac{f^{4)}(\xi)}{4!}\Pi(x)\right| \le \frac{0.375}{4!}0.084 = \underline{0.0013125 \ge |e(x)| \quad \forall x \in [2,3]}$$

E)

To approximately minimize the maximum value of $|e(x)|$ over the whole interval $[2, 3]$, we need to use Chebyshev nodes. For the degree of the polynomial to be ≤ 3 we need four nodes. We first have to calculate the four roots t_i of the Chebyshev polynomial of the first kind of degree 4:

$$\cos(4\theta_i) = 0 \quad\Rightarrow\quad 4\theta_i = \pi/2 + k\pi \quad\rightarrow\quad t_i = \cos(\theta_i) = \cos\left(\frac{\pi/2 + k\pi}{4}\right) \quad (k = 0,1,2,3)$$

With Octave:

```
k = 0:3; ti = cos((pi/2 + k*pi)/4), ti = anm_signif(sort(ti), 5)
     ti = 0.92387953251129  0.3826834323651  -0.3826834323651  -0.92387953251129
     ti =         -0.92388         -0.38268           0.38268            0.92388
```

Now we have to apply the linear transformation from $[-1, 1]$ to $[a, b] = [2, 3]$:

$$x = \frac{a+b}{2} + \frac{b-a}{2} t_i = \frac{3+2}{2} + \frac{3-2}{2} t_i = 2.5 + 0.5 t_i$$

```
xi = 2.5 + 0.5 * ti, xi = anm_signif(xi, 5)
     xi =          2.03806          2.30866          2.69134          2.96194
     xi =          2.0381           2.3087           2.6913           2.9619
fi = anm_signif(f(xi), 5)
     fi =          0.71202          0.83668          0.99002          1.0858
```

So the interpolation points are:

$$\underline{(2.0381, 0.71202),\quad (2.3087, 0.83668),\quad (2.6913, 0.99002),\quad (2.9619, 1.0858)}$$

F)

The error of $p_b(2.55)$ with respect to $f(2.55)$ is:

$$\log(2.55) - p_b(2.55) = 0.93609 - 0.93614 = -5\times10^{-5}$$

This error is quite larger (in absolute value) than the error of p_a at the same point, which, remember, was estimated to be about -7×10^{-7} (and more precisely -3×10^{-7}). Not too good for a polynomial, p_b, which is supposed to minimize $|e(x)|$ in some sense. But this result should not be surprising, because p_b minimizes (approximately) the *maximum* value of $|e(x)|$ *in an interval*, not at any given point.

The conclusion we can draw is that p_b is not particularly good at estimating $f(2.55)$, but neither is it particularly bad at estimating f at any point in $[2, 3]$. We can be very sure that there will be many points in $[2, 3]$ where the error made by p_a will be much larger than absolutely any error made by p_b at any point of the same interval.

G)

Most of the work was done in section D). The only difference now is the upper bound of $|\Pi(x)|$. But in this case, with Chebyshev nodes in $[2, 3]$, we know that the maximum absolute value of Π is reached at the interval endpoints (and once between each two successive nodes). The easiest is to evaluate Π at $x = 2$ or at $x = 3$:

```
xi                                  % make sure xi still has the Chebyshev nodes
     xi =       2.0381       2.3087       2.6913       2.9619
prod(2 - xi), prod(3 - xi)
     ans =       0.0078209243805609
     ans =       0.0078209243805609
```

Now like in section D):

$$|e(x)| = \left| \frac{f^{4)}(\xi)}{4!} \Pi(x) \right| \le \frac{0.375}{4!} 0.0078209 = \underline{0.00012220 \ge |e(x)| \quad \forall x \in [2,3]}$$

This is, as expected, far less than 0.0013125, which is what we found for p_a.

EXERCISE 2 **(10 points)**

Observing the symmetry in the numbers in the table we can integrate over just a quarter of the region. This does not mean not using all the information in the table, but using it wisely.

The second Simpson rule needs 4 nodes, which discards its use in the x direction, for which we have 5 nodes for the whole interval or 3 for half of it. But the compound trapezoidal rule can be used with any number of nodes (even when they are not equally spaced, actually).

In the y direction we can use the compound Second Simpson rule with two subintervals, if we don't notice the symmetry (nodes 1st to 4th, and nodes 4th to 7th), or the simple rule between $y = 0$ and $y = 0.6$, if we do.

In the *x* direction:

The simple Trapezoidal rule reads: $Q_{x,S} = \frac{h_x}{2}(f_0 + f_1)$

Therefore the compound rule with 2 subintervals (to reach $x = 0.8$) is:

$$Q_{x,C} = \frac{h_x}{2}(f_0 + f_1) + \frac{h_x}{2}(f_1 + f_2) = \frac{h_x}{2} f_0 + h_x f_1 + \frac{h_x}{2} f_2$$

So the *x*-weights are: $w_{x0} = h_x/2$; $w_{x1} = h_x$; $w_{x2} = h_x/2$

With $h_x = 0.4$: $w_{x0} = 0.2$; $w_{x1} = 0.4$; $w_{x2} = 0.2$

In the *y* direction:

If we notice the symmetry, we only need the simple rule, whose weights (provided) are:

$$w_{y0} = 3h_y/8; \quad w_{y1} = 9h_y/8; \quad w_{y2} = 9h_y/8; \quad w_{y3} = 3h_y/8$$

With $h_y = 0.2$: $w_{y0} = 0.075$; $w_{y1} = 0.225$; $w_{y2} = 0.225$; $w_{y3} = 0.075$

2D composition of the weights:

For each 2D node, the resulting weight is the product of its weights in the *x* and in the *y* directions.

With Octave:

```
hx = 0.4
     hx =      0.4
wxi = [hx/2 hx hx/2]
     wxi =     0.2          0.4          0.2
hy = 0.2
     hy =      0.2
wyj = 3/8*[hy 3*hy 3*hy hy]
     wyj =   0.075        0.225        0.225        0.075
```

Now observe this trick (just a matrix multiplication of a column by a row):

```
wij = wxi' * wyj              % 2D weights
     wij =   0.015        0.045        0.045        0.015
              0.03         0.09         0.09         0.03
             0.015        0.045        0.045        0.015
```

Corresponding 2D nodal values (copied from the table provided):

```
fij = [0 0 0 0; 0 3.123 4.794 5.319; 0 3.818 5.960 6.647]
     fij =       0            0            0            0
                 0        3.123        4.794        5.319
                 0        3.818         5.96        6.647
qij = wij .* fij              % elementwise product
     qij =       0            0            0            0
                 0      0.28107      0.43146      0.15957
                 0      0.17181       0.2682     0.099705
Q = sum(sum(qij))             % final result in 1st quarter
     Q =    1.4118
Q = 4 * Q                     % don't forget the symmetry applied
     Q =    5.6473
```

Therefore $S \approx 5.647$ units of effort

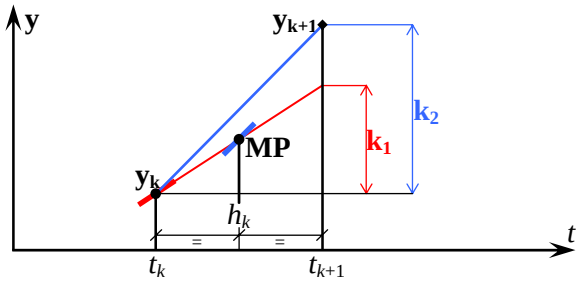
EXERCISE 3 **(12 points)**

A)

We first need the advance formula of the Midpoint method (or Modified Euler method), which is:

$$\begin{cases} \mathbf{k_1} = \mathbf{f}(t_k, \mathbf{y_k}) h_k \\ \mathbf{k_2} = \mathbf{f}(t_k + h_k/2, \mathbf{y_k} + \mathbf{k_1}/2) h_k \\ \mathbf{y_{k+1}} = \mathbf{y_k} + \mathbf{k_2} \end{cases}$$

This formula is very natural and easy to remember with just a little mental graph of the method's advance scheme, which is represented here:



The idea is to first take a half step with slope equal to the value of **f** at the point of departure $(t_k,\mathbf{y_k})$, evaluate **f** at the intermediate point of arrival (midpoint **MP** in the figure), and finally use that slope, which seems a reasonable average for the whole step, to move from $(t_k,\mathbf{y_k})$ to $(t_{k+1},\mathbf{y_{k+1}})$.

The numeric application of this formula will be done with Octave. You should be able to reproduce these operations with your calculator. The commands are devoid of any initial >> characters so you can also copy-paste them into Octave easily, if you feel so inclined:

```
format long g                          % to see more digits on screen
f = @(t,y) -0.06 * sqrt(y);            % ODE right-hand side (since k = 0.06)
h = 0.25; t0 = 0; t1 = h; t2 = 2*h;    % abscissas needed
y0 = 3;                                % initial condition
k1 = f(t0, y0) * h                     % apply advance formula...
     k1 =   -0.0259807621135332
k2 = f(t0 + h/2, y0 + k1/2) * h
     k2 =   -0.0259244510889281
y1 = y0 + k2                           % first step finished
     y1 =      2.97407554891107
k1 = f(t1, y1) * h                     % overwriting k1
     k1 =   -0.0258682623789266
k2 = f(t1 + h/2, y1 + k1/2) * h        % overwriting k2
     k2 =   -0.0258119510883498
y2 = y1 + k2                           % second step finished
     y2 =      2.94826359782272
[tout, yout] = anm_ode(f, [0 0.5], y0, 2, 'Midpoint')    % double check
     tout =    0                  0.25                  0.5
     yout =    3      2.97407554891107      2.94826359782272
```

All seems good. You can use fewer significant digits (what matters is that the process is carried out correctly). The solution is: $y(0.25) \approx 2.9741$, $y(0.5) \approx 2.9483$

B)

Exact solution at $t = 30$:

$$y(30) = 3 - 0.103923 \cdot 30 + 0.0009 \cdot 30^2 = 0.69231$$

Global error at $t = 30$ with $h = 0.5$:

$$E_{0.5} = 0.69231 - 0.6831 = 0.00921$$

Global error at $t = 30$ with $h = 0.25$:

$$E_{0.25} = 0.69231 - 0.6877 = 0.00461$$

Global error at $t = 30$ with $h = 0.125$:

$$E_{0.125} = 0.69231 - 0.6900 = 0.00231$$

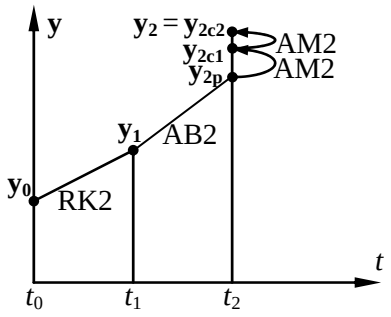
Euler's method is of order of convergence 1, so when the step size h is halved, and if h is small enough, we expect the error to be also halved. Let's see if that is the case:

$$E_{0.25} / E_{0.5} = 0.00461 / 0.00921 \approx 0.5005$$
$$E_{0.125} / E_{0.25} = 0.00231 / 0.00461 \approx 0.501$$

Both are close to 0.5, so this agrees well with the expected behavior.

C)

To start a multistep method (or a predictor-corrector using multistep methods) we usually need more points than just the initial conditions. That's why we are told to "start" the method with an RK2 method (of the same order 2 as the predictor-corrector). In our case the AB2 method needs 2 calculated points (see its advance formula, which is the explicit one of the two ones provided), so we need to take *one* step of an RK2 method. After that, the AB2 method will do the predictions and the AM2 method will do the corrections, and we won't need the RK2 any more. This scheme illustrates what we have to do:



The Midpoint or Modified Euler method is an instance of Runge-Kutta method of order 2 (like also the Heun or Enhanced Euler), so we can take advantage of the work done in section A). Again indicating the operations with Octave:

```
h, t0, t1, y0, y1, f                % from having done the Midpoint method before
     h =                     0.25
     t0 =                       0
     t1 =                    0.25
     y0 =                       3
     y1 =       2.97407554891107
     f = @(t,y) -0.06 * sqrt(y)
f0 = f(t0, y0)
     f0 =    -0.103923048454133
f1 = f(t1, y1)
     f1 =    -0.103473049515707
y2p = y1 + h/2 * (3 * f1 - f0)    % y2 "predicted"
     y2p =     2.94826353639945
f2p = f(t2, y2p)
     f2p =   -0.103023049513388
y2c1 = y1 + h/2 * (f1 + f2p)      % y2 "corrected once"
     y2c1 =    2.94826353653243
f2c1 = f(t2, y2c1)
     f2c1 =  -0.103023049515712
y2c2 = y1 + h/2 * (f1 + f2c1)     % y2 "corrected twice"
     y2c2 =    2.94826353653214
y2 = y2c2                         % take as definitive (two corrections)
     y2 =      2.94826353653214
```

Hence: $\underline{y(0.25) \approx 2.9741, \quad y(0.5) \approx 2.9483}$

The difference in y_2 with the result using the Midpoint method is not seen until several digits later.

D)

When a method to solve initial-value problems is unstable, it does not often amplify errors or perturbations in a subtle or controlled way. Most of the times the perturbations, amplified over many steps, make the numerical results totally useless and different from the exact solutions being searched. You will detect numerical instability because of the "crazy" results of your program, often including lots of "**`Inf`**" (floating-point infinity) elements.

The first step to deal with this should be to decrease the step size *h*. If your ODE or system thereof is stable, a sufficiently small value of *h* will always also make your method stable.

Sometimes decreasing *h* enough is not feasible because of the computational cost it entails. This can happen especially with very stiff systems of ODEs, which force one to use extremely small values of *h* before "normal" methods can be stable. In that case one can use methods specially indicated to solve stiff systems, in that they have very ample stability regions. Implicit multistep methods and their predictor-corrector counterparts may fall into this category.

Finally, if everything else fails, one can use unconditionally stable methods (A-stable methods), whose absolute stability region is the whole complex left half-plane. This means that no matter how stiff the system of ODEs is (as long as it is stable), the method will also be stable with any step size. The problem is that their orders of convergence are very limited (only 1 and 2). The most well-known unconditionally stable methods are the Backward Euler and the Trapezoidal method. The former is of order 1, while the latter is of order 2, so it would be the one of choice. Of course the problem with implicit methods is that they require to carry out iterations on every step in order to advance. This increases the computational cost (theoretically to infinity, if one tries to satisfy the advance formula exactly). In practice you will settle with a finite number of iterations, i.e. you will apply a predictor-corrector method with corrections made with the Trapezoidal method (like the Heun predictor-corrector method).

EXERCISE 4 **(7 points)**

A)

We can do this via indeterminate coefficients, or differentiating Lagrange base polynomials, or using Taylor series… For instance, via indeterminate coefficients, we can impose the exact differentiation of $f(x) = 1, x, x^2, x^3$ (see the theory to know why). We can already make $z = 0$ because the coefficients will not depend on *z*:

$$\begin{aligned}
&f(x)\equiv 1: \quad f''(0) = 0 = A_0 f(x_0) + A_1 f(x_1) + A_2 f(x_2) + A_3 f(x_3) \quad \Rightarrow \quad A_0 + A_1 + A_2 + A_3 = 0\\
&f(x) = x: \quad f''(0) = 0 = A_0 (-2h) + A_1 (-h) + A_2 h + A_3 2h \quad \Rightarrow \quad -2A_0 - A_1 + A_2 + 2A_3 = 0\\
&f(x) = x^2: \quad f''(0) = 2 = A_0 (-2h)^2 + A_1 (-h)^2 + A_2 h^2 + A_3 (2h)^2 \quad \Rightarrow \quad 4A_0 + A_1 + A_2 + 4A_3 = 2/h^2\\
&f(x) = x^3: \quad f''(0) = 6\cdot 0 = A_0 (-2h)^3 + A_1 (-h)^3 + A_2 h^3 + A_3 (2h)^3 \quad \Rightarrow \quad -8A_0 - A_1 + A_2 + 8A_3 = 0
\end{aligned}$$

A good way to solve this system by hand is to apply Gauss's method on its augmented matrix:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ -2 & -1 & 1 & 2 & 0 \\ 4 & 1 & 1 & 4 & 2/h^2 \\ -8 & -1 & 1 & 8 & 0 \end{pmatrix} \begin{matrix} \\ <R_2 + 2R_1> \\ <R_3 - 4R_1> \\ <R_4 + 8R_1> \end{matrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 3 & 4 & 0 \\ 0 & -3 & -3 & 0 & 2/h^2 \\ 0 & 7 & 9 & 16 & 0 \end{pmatrix} \begin{matrix} \\ \\ <R_3 + 3R_2> \\ <R_4 - 7R_2> \end{matrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 3 & 4 & 0 \\ 0 & 0 & 6 & 12 & 2/h^2 \\ 0 & 0 & -12 & -12 & 0 \end{pmatrix} \begin{matrix} \\ \\ \\ <R_4 + 2R_3> \end{matrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 3 & 4 & 0 \\ 0 & 0 & 6 & 12 & 2/h^2 \\ 0 & 0 & 0 & 12 & 4/h^2 \end{pmatrix}$$

Backward substitution: $A_3 = \dfrac{4/h^2}{12} = \dfrac{1}{3h^2} \quad \rightarrow \quad A_2 = \dfrac{1}{6}\left(\dfrac{2}{h^2} - \dfrac{4}{h^2}\right) = \dfrac{-1}{3h^2}$

$$\rightarrow \quad A_1 = \frac{3}{3h^2} - \frac{4}{3h^2} = \frac{-1}{3h^2} \quad \rightarrow \quad A_0 = -\frac{1}{3h^2} - \frac{-1}{3h^2} - \frac{-1}{3h^2} = \frac{1}{3h^2}$$

Hence:

$$\boxed{f''(z) \approx D = \frac{f(z-2h) - f(z-h) - f(z+h) + f(z+2h)}{3h^2}}$$

B)

These asymptotic expansions of the error term are easy to obtain (albeit tedious) using Taylor series expansions by increasing the maximum order in the error term as desired (i.e. increasing *m* in the

theory we have studied). Given my lazy nature, I would not do it by hand, but program the computer to do it for me, because the operations involved are very systematic.

C)

E has derivatives of f only of orders ≥ 4. The formula will be exact in $\mathbb{P}_3$ because every derivative of order ≥ 4 of a polynomial of degree ≤ 3 is identically zero, and hence $E = 0$. However, x^4 has its fourth derivative non-zero everywhere (it is $4! = 24$) but null derivatives or orders ≥ 6, so for $f(x) = x^4$, $E = -5/12 \cdot 24\, h^2 \neq 0$, and the formula will not be exact in $\mathbb{P}_4$. Hence the polynomial degree is $N = 3$ (which means that the formula is exact with every polynomial of degrees ≤ 3, but not ≥ 4).

The order of convergence (order of precision) is $O = 2$, because that is the exponent of h in the principal term of E (the 1st one, because it has the lowest exponent of h, so it will be much larger than all the other terms when h is small enough). $O = 2$ means that as h tends to 0, E will tend to zero approximately as a constant times h^2 (or faster, if $f^{4)}(z) = 0$). More formally, it means that $\exists\, k, h_0 \in \mathbb{R}^+ \,/\, \forall h$ with $|h| \leq h_0$, $|E| \leq k\, h^2$.

D)

We are told to consider $E \approx -5/12\, f^{4)}(z)\, h^2$. This is very reasonable for even moderately small values of h. For instance, if only $h = 10^{-6}$, the value of h^2 in the firs term is 10^{-12}, while the value of h^4 in the second term is 10^{-24}. Unless the derivatives of f "explode" very fast (and in that case a smaller value of h will do), the first term will be much larger (in absolute value) than all the other ones combined.

Abusing notation for convenience, we will use = instead of ≈ for a while:

$$|E_{tot}| \leq |E| + |E_r| \quad \text{where} \quad |E| \leq g_1(h), \quad |E_r| \leq g_2(h), \quad \text{where:}$$

$$g_1(h) = 5/12\, M\, h^2 \quad \text{where} \quad M = |f^{4)}(z)|;$$

$$\text{Amplification factor} = AF = \text{sum}(|A_i|) = (1/3 + 1/3 + 1/3 + 1/3) / h^2 = 4/3\, h^{-2}$$

$$g_2(h) = AF\, \varepsilon = 4/3\, h^{-2}\, \varepsilon \quad \text{where} \quad \varepsilon \geq |f_i - \overline{f}_i| \;\forall\; i = 0,...,n.$$

$$|E_{tot}| \leq g_1(h) + g_2(h) = g(h) = 5/12\, M\, h^2 + 4/3\, h^{-2}\, \varepsilon$$

$$h_{opt} \quad \Rightarrow \quad g \text{ min} \quad \Rightarrow \quad g'(h) = 2 \cdot 5/12\, M\, h - 2 \cdot 4/3\, h^{-3} \varepsilon = 0 \quad \Rightarrow$$

$$h_{opt} = (16/5\, \varepsilon/M)^{1/4} = 1.3374806099529\, \varepsilon^{1/4}\, M^{-1/4}$$

Hence:

$$\boxed{h_{opt} \approx \sqrt[4]{\frac{16}{5}\frac{\varepsilon}{M}} \approx 1.3375\, \varepsilon^{1/4} M^{-1/4}}$$

where we use again ≈ instead of = to end the abuse of notation.

Here, as stated above, ε is an upper bound of the absolute-value error in the nodal ordinates, which is usually linked to the precision of the arithmetic we are using:

$$\varepsilon \geq \left| f(x_i) - \overline{f}(x_i) \right| \quad (\forall i = 0,\ldots,n)$$

and M is the absolute value of $f^{4)}$ at the point z where we use the formula $(M = |f^{4)}(z)|)$, or it can also be an upper bound of $f^{4)}$ at the points z where we want to use it: $M \geq \left| f^{4)}(z) \right|$.