ADVANCED NUMERICAL METHODS – EXAM OF 6/23/2015 FULL ANSWER

N.B. The answers here are far more complete than could be possibly expected from the student in a real exam situation. Their main purpose is a teaching one, not an assessment-related one.

Exercise 1

a)

To minimize $\|e(x)\|_\infty$ in the interval exactly, the student would have to use ad-hoc nodes for the specific function $f(x)$ she is interpolating. And this answer is perfectly ok. ■

But if she is content with minimizing that error quite approximately —and if she has studied the theory— the student will use the Chebyshev nodes. She will first look for the ones in the interval $[-1,1]$, which are the roots of the Chebyshev polynomial of the 1st kind of degree 5; and then apply the linear mapping onto $[1.74, 6.26]$:

$$T_5(t) = \cos(5\arccos(t)) = 0 \quad \Rightarrow \quad 5\arccos(t) = \pi/2 + k\pi \quad \Rightarrow$$

$$\arccos(t) = \frac{\pi/2 + k\pi}{5} \quad \rightarrow \quad t_i = \cos\left(\frac{\pi/2 + k\pi}{5}\right) \quad (k = 0,1,2,3,4)$$

$$x_i = \frac{1.74 + 6.26}{2} + \frac{6.26 - 1.74}{2} t_i = 4 + 2.26 t_i$$

Using Matlab as our calculator (and rounding to 3 significant digits, albeit without full diligence):

```
>> k = 0:4; ti = cos((pi/2+k*pi)/5)
ti =  0.95106      0.58779  6.1232e-017     -0.58779     -0.95106
>> % won't affect, but central 6.1e-017 is clearly 0 with a roundoff error
>> xi = 4 + 2.26*ti, xi = anm_signif(xi(end:-1:1),3)
xi =   6.1494       5.3284            4       2.6716       1.8506
xi =     1.85         2.67            4         5.33         6.15
```

■

An additional question that could have been asked is under what conditions $\|e(x)\|_\infty$ is minimized *exactly* in the interval by using these nodes. To answer it, we look at this expression of the error term:

$$e(x) = \frac{f^{n+1)}(\xi)}{(n+1)!}\Pi(x)$$

We know that the Chebyshev nodes minimize $\|\Pi(x)\|_\infty$; but if $f^{n+1)}$ fluctuates, the minimization of $\|e(x)\|_\infty$ is typically not attained (or is attained only approximately, if $f^{n+1)}$ does not fluctuate too much). To guarantee the exact minimization of $\|e(x)\|_\infty$ it is sufficient (although not necessary) that $f^{n+1)}$ is constant, i.e. f a polynomial of degree $n+1$ (or, trivially, less than $n+1$, with identically zero error).

b)

Since the nodes are unevenly spaced, we must use divided differences. Operating *diligently* with 3 significant digits (minimum precision required) these are:

```
>> (0.309-0.844)/(2.4-1.82), signif(ans,3)
ans =  -0.92241
ans =  -0.922
>> (-0.249-0.309)/(3.42-2.4), signif(ans,3)
ans =  -0.54706
ans =  -0.547
>> (-0.547 - -0.922)/(3.42-1.82), signif(ans,3)
ans =   0.23438
ans =   0.234
>> (0 - -0.547)/(4.58-2.4), signif(ans,3)
ans =   0.25092
ans =   0.251
>> (0.251-0.234)/(4.58-1.82), signif(ans,3)
ans =   0.0061594
ans =   0.00616
>> (0-0.00616)/(5.6-1.82), signif(ans,3)
ans =  -0.0016296
ans =  -0.00163
```

and a couple very similar numbers. The resulting table is:

i	xi	fi0	fi1	fi2	fi3	fi4	fi5	
0	1.82	0.844						
1	2.4	0.309	-0.922					
2	**3.42**	**-0.249**	-0.547	0.234				
3	**4.58**	-0.249	**0**	0.251	0.00616			
4	**5.6**	0.309	0.547	**0.251**	0	-0.00163		
5	6.18	0.844	0.922	0.234	**-0.00616**	-0.00163	0	■

With more precise arithmetic (e.g. Matlab's double precision one) the numbers do not change much.

Here the additional node seems to provide redundant information only, because the last divided difference is zero, so the (unique) interpolation polynomial of degree ≤ 5 is of degree 4. Hence, the polynomial by any 5 of the 6 nodes will be the same as the polynomial by all 6 nodes, so any of the 6 nodes could be the "unnecessary" one. But note that providing redundant information is in real life very different from providing no information at all. In our case the additional node suggests that the function $f(x)$ the data come from may be[1] a polynomial of degree 4, which is something we would not suspect with only 5 nodes. (On another note, observing the table suggests very strongly that $f(x)$ is symmetric on both sides of the interval's midpoint, with only three interpolation points being truly independent.)

c)
With the numbers in the table of divided differences in bold:

$$p_3(x) = -0.249 + 0 + 0.251(x-3.42)(x-4.58) - 0.00616(x-3.42)(x-4.58)(x-5.6) \quad ■$$

The last four interpolation points are indeed the ones of choice for $p_3(x)$ in this case, because they are the closest ones to the point $x = 5.1$ where the polynomial will be evaluated, with two nodes on each side of x. In this way the polynomial oscillations (Runge effect), smaller near the center of the nodal interval than its ends, will be lessened, and even the roundoff errors will probably be reduced.

[1] It actually isn't.

d)
Substituting:

$$p_3(5.1) = -0.249 + 0.251(5.1-3.42)(5.1-4.58) - 0.00616(5.1-3.42)(5.1-4.58)(5.1-5.6) =$$
$$= -0.249 + 0.251\times1.68\times0.52 + 0.00616\times1.68\times0.52\times0.5$$

The Hörner-like algorithm consists basically in taking as many common factors as possible:

$$p_3(5.1) = -0.249 + 1.68\times0.52\times[0.251 + \underbrace{\underbrace{0.00616\times0.5}_{0.00308}}_{0.25408\to0.254}]_{0.13208\to0.132}]_{0.22176\to0.222}]_{-0.0270} = \boxed{p_3(5.1) = -0.0270 \approx f(5.1)}$$ ■

Thanks to the common factors, this way to evaluate polynomials is optimal in the sense of requiring the least number of arithmetic operations. It also behaves well in terms of roundoff error propagation and can be efficiently implemented in a computer in terms of memory storage.

e)
The Newton polynomial of degree n (or less), $p_n(x)$, is:

$$p_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \ldots + a_{n-1}(x-x_0)(x-x_1)\cdots(x-x_{n-2}) + a_n(x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

and of course the one of degree n+1 (or less), $p_{n+1}(x)$, is the same thing with n+1 instead of n.

The divided difference $f[x_0, x_1, \ldots, x_n]$ can be defined as the leading coefficient of $p_n(x)$. In Newton's representation, that is its last coefficient:

$$f[x_0, x_1, \ldots, x_n] = a_n$$

Calling $h_n(x)$ the last term of $p_n(x)$:

$$h_n(x) = f[x_0, x_1, \ldots, x_n](x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

and hence: $$h_{n+1}(x) = f[x_0, x_1, \ldots, x_n, x_{n+1}](x-x_0)(x-x_1)\cdots(x-x_n)$$

which is also equal to $p_{n+1}(x) - p_n(x)$, or the term that must be added to $p_n(x)$ in order to obtain $p_{n+1}(x)$.

Now calling as usual $$\Pi(x) = (x-x_0)(x-x_1)\cdots(x-x_{n-1})(x-x_n)$$

we have: $$h_{n+1}(x) = p_{n+1}(x) - p_n(x) = f[x_0, x_1, \ldots, x_n, x_{n+1}]\,\Pi(x)$$

To *obtain* (exact formula) the error $e(x)$ made by the interpolation polynomial p_n at a point x, consider adding a new interpolation point $(x_{n+1}, f_{n+1}) = (x, f(x))$ and calculating the corresponding interpolation polynomial p_{n+1}. By definition $p_{n+1}(x) = f(x)$, so the error made by p_n on x will be:

$$e(x) = f(x) - p_n(x) = p_{n+1}(x) - p_n(x) = h_{n+1}(x) = f[x_0, x_1, \ldots, x_n, x_{n+1}]\,\Pi(x)$$

and since $x_{n+1} = x$: $$e(x) = f[x_0, x_1, \ldots, x_n, x]\,\Pi(x)$$

as we wanted to prove (with x in place of z). ■

f)
The expression we just proved is *exact* (if the arithmetic too is) because that is what the symbol "=" means in Mathematics. However, it can be used to *estimate* the error $e(x)$ if we assume that the unknown[2] divided difference appearing in it, $f[x_0, x_1, \ldots, x_n, x]$, is close to some other difference of the same order at our disposal, for instance $f[x_0, x_1, \ldots, x_n, x_{n+1}]$, where x_{n+1} is a new node. Hence, if one has a new interpolation point (x_{n+1}, f_{n+1}) available, the error of $p_n(x)$ can be estimated by the expression:

[2] It could be known if we knew $f(x)$, but then the error $e(x)$ would simply and exactly be equal to $f(x) - p_n(x)$, without the need to estimate anything.

$$e(x) \approx f[x_0, x_1, \ldots, x_n, x_{n+1}]\,\Pi(x)$$

In our case: $e(x) = f[x_2, x_3, x_4, x_5, x]\,\Pi(x) \quad \Rightarrow \quad e(5.1) = f[3.42, 4.58, 5.6, 6.18, 5.1]\,\Pi(5.1)$

and, in place of the unknown fourth-order difference appearing here, we will use, assuming that both are close, the one that can be seen, even by duplicate (which seems to confer it some "credibility"), in the table of divided differences, namely –0.00163. Evaluating at $x = 5.1$ and substituting values:

$$e(5.1) \approx -0.00163\,\Pi(5.1) = -0.00163\,(5.1-x_2)(5.1-x_3)(5.1-x_4)(5.1-x_5) =$$

$$= -0.00163\,(5.1-3.42)(5.1-4.58)(5.1-5.6)(5.1-6.18) = \boxed{-0.000769} \qquad \blacksquare$$

With double-precision arithmetic throughout the value obtained is –0.000739 instead of –0.000769. At any rate, it's a very small one. If true, it would mean that $p_3(5.1) = -0.0270$ is a very good approximation to $f(5.1)$, which should be somewhere between –0.0277 and –0.0278.

Substituting, as we did, a divided difference of order $n+1$ by another is often reasonable, because $f[x_0, x_1, \ldots, x_n, x] = f^{n+1)}(\xi_1)/(n+1)!$ for some value ξ_1 between the points $x_0, x_1, \ldots, x_n, x$, while $f[x_0, x_1, \ldots, x_n, x_{n+1}] = f^{n+1)}(\xi_2)/(n+1)!$ for some value ξ_2 between the points $x_0, x_1, \ldots, x_n, x_{n+1}$; and, although in general $\xi_1 \neq \xi_2$, if $f^{n+1)}$ behaves "well" (it does not undergo large fluctuations nor gets very close to zero), it is often acceptable to consider $f^{n+1)}(\xi_1) \,/\, f^{n+1)}(\xi_2) \approx 1$.

Another way to estimate $e(5.1)$ is as $p_{n+1}(5.1)$ (supposedly our presently best estimation of $f(5.1)$) minus $p_n(5.1)$. The result is the same, because this is actually a different way to justify the same operations. In general, if the error made by $p_n(x)$ is the exact minus the approximate value ($e(x) = f(x) - p_n(x)$), then our best estimation of that error should be equal to our {best estimation of $f(x)$} minus $p_n(x)$. Assuming that the new interpolation point (x_{n+1}, f_{n+1}) improves the precision of the interpolation polynomial at x, our best estimation of $f(x)$ becomes now $p_{n+1}(x)$, and our best estimation of the error made by $p_n(x)$ becomes $p_{n+1}(x) - p_n(x)$. But that is precisely $h_{n+1}(x)$:

$$e(x) \approx h_{n+1}(x) = f[x_0, x_1, \ldots, x_n, x_{n+1}]\,\Pi(x)$$

as before. In this case, under the assumption that $p_{n+1}(x)$ is close to $f(x)$. This reasoning has an advantage over the one based on substituting one divided difference by another, namely, that we can apply it unequivocally when we have more than one extra interpolation point. It is also a clear reasoning.

Finally note that sometimes assuming that $f^{n+1)}$ does not undergo large variations nor gets close to 0, or assuming that $f[x_0, x_1, \ldots, x_n, x] \approx f[x_0, x_1, \ldots, x_n, x_{n+1}]$, or assuming that $p_{n+1}(x) \approx f(x)$, is not correct, and the estimation of the error $e(x)$ fails, even spectacularly.

g)

The error actually made in d) is:

$$e(5.1) = f(5.1) - p_3(5.1) = \cos(\pi \times 5.1) - -0.0270 = -0.951 + 0.0270 = \boxed{-0.924}$$

This is not even close to –0.000769 as estimated in f). It seems I was too optimistic. The error is actually $0.924/0.000769 \approx 1200$ times larger than expected. Neither is the exact value $f(5.1) = -0.951$ close to being between –0.0277 and –0.0278 as I had predicted. I am getting nervous. $\blacksquare$

To investigate what can be happening I will start by checking that the numbers in the table provided indeed correspond to the six Chebyshev nodes the student wanted to use in $[1.74, 6.26]$, and that the ordinates indeed correspond to the function $f(x) = \cos(\pi x)$. Using Matlab:

```
>> k = 0:5; ti = cos((pi/2+k*pi)/6); ti = ti(end:-1:1);
>> xi = 4 + 2.26*ti; xi = signif(xi,3);
>> fi = cos(pi*xi);  fi = signif(fi,3); xi, fi
xi =    1.82        2.4       3.42       4.58        5.6       6.18
fi =   0.844      0.309     -0.249     -0.249      0.309      0.844
```

The numbers coincide exactly…

So maybe the problem has to do with the very-low-precision arithmetic we are using? Three significant digits is very little… However, doing everything with double-precision arithmetic, nothing changes by much. This does not have to do with roundoff errors.

I will now try to see if assimilating $f[3.42, 4.58, 5.6, 6.18, 5.1]$ to -0.00163 was a good idea (although, in light of what we already know, it can't have been). To obtain $f[3.42, 4.58, 5.6, 6.18, 5.1]$ we can reuse most of the table of differences of section b), adding just one more final row (in bold):

```
i      xi       fi0       fi1       fi2         fi3         fi4
2    3.42    -0.249    -0.547    0.234
3    4.58    -0.249         0    0.251     0.00616
4     5.6     0.309     0.547    0.251           0    -0.00163
5    6.18     0.844     0.922    0.234    -0.00616    -0.00163
6     5.1    -0.951      1.66    -1.48        -3.3       -1.96
```

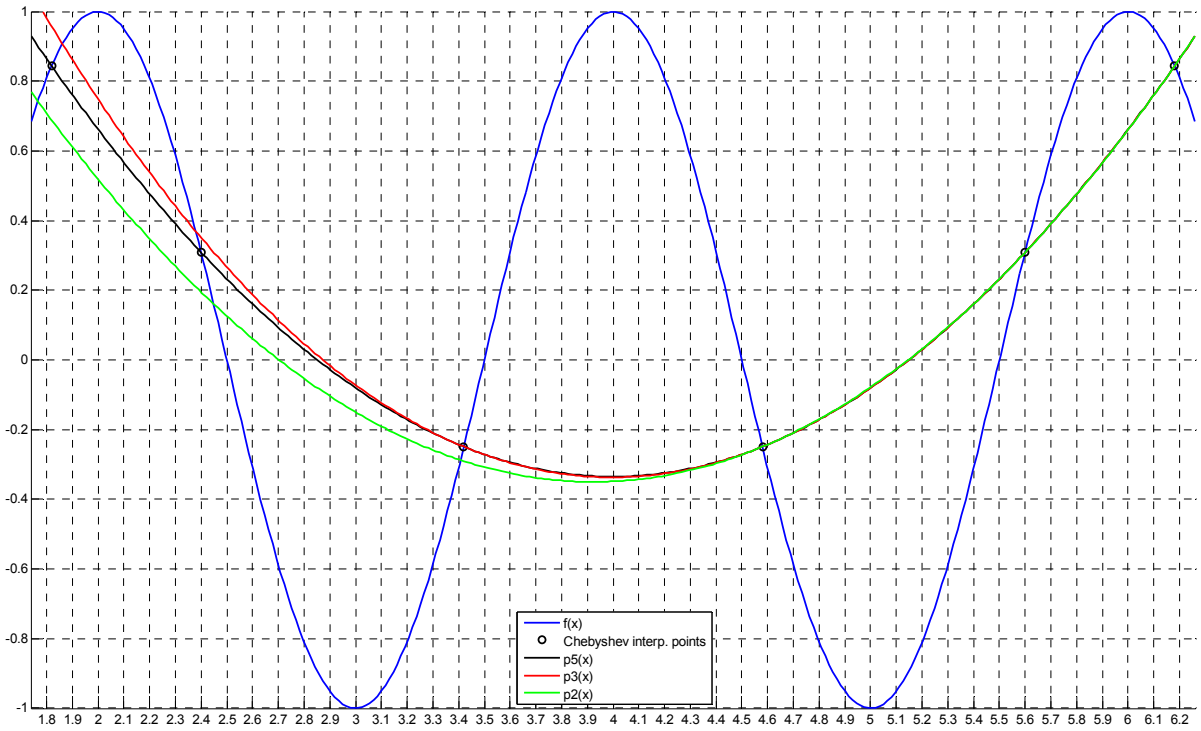
These are the operations:

```
>> (-0.951-0.844)/(5.1-6.18), signif(ans,3)
ans =   1.662
ans =   1.66
>> (1.66-0.922)/(5.1-5.6), signif(ans,3)
ans =  -1.476
ans =  -1.48
>> (-1.48-0.234)/(5.1-4.58), signif(ans,3)
ans =  -3.2962
ans =  -3.3
>> (-3.3 - -0.00616)/(5.1-3.42), signif(ans,3)
ans =  -1.9606
ans =  -1.96
```

Hence $f[3.42, 4.58, 5.6, 6.18, 5.1] = -1.96$, which is nowhere close to -0.00163. The ratio, $-1.94/-0.00163 \approx 1200$, logically coincides with the one existing between the real and estimated errors. The idea to substitute one divided difference by another has been catastrophic.

But why are both differences so different? The figure below helps give some understanding. On the one hand, the nodes chosen, even if they are indeed the Chebyshev ones, are too separated for the variability exhibited by $f(x)$. And additionally it turns out that the 6 Chebyshev nodes in the interval $[1.74, 6.26]$, when applied to the function $f(x) = \cos(\pi x)$, result in a set of interpolation points that are almost exactly aligned in a polynomial of degree 3 (even 2), even if the nature of $f(x)$ has nothing to do with a low-degree polynomial. This is pure coincidence[3]. The polynomials $p_5(x) = p_4(x)$ and $p_3(x)$ (and even $p_2(x)$, using each time one less node from the right) take values at $x = 5.1$ that are extremely close to one another. This even seems to give confidence in those values. However, none of them is close to $f(5.1)$. Without knowing that $f(x) = \cos(\pi x)$, nothing could have made us suspect this ($f(x)$ could have perfectly been a low-degree polynomial!)

[3] The coincidence is actually not pure, but could have been.



Exercise 2

a)

Let us first check the data in the table:

```
>> a=1; b=3; M=4; h=(b-a)/M/2; xi = a+h:2*h:b-h  % check given nodes
xi =     1.25         1.75         2.25         2.75
>> f = @(x) exp(-x.^2).*(x.^2+1);                % use f(x) defined in d)
>> fi = f(xi); fi = round(fi*1e4)/1e4            % check given ordinates
fi =   0.5371         0.19       0.0384       0.0044
```

Looks good. Apply the compound Midpoint rule with $M=4$ subintervals:

```
>> Qa = 2*h*sum(fi)                             % Qa means "Q for part a)"
Qa =                                  0.38495
```

b)

Substituting: $$|E| = \left|\frac{b-a}{6}h^2 f''(\xi)\right| \le \frac{3-1}{6}0.25^2 \times 0.89617 = \boxed{0.01867}$$

c)

We are after a sufficient but not necessary condition, so the "⇐" symbol will appear:

$$E = \frac{b-a}{6}h^2 f''(\xi) = \frac{b-a}{6}\left(\frac{b-a}{2M}\right)^2 f''(\xi) = \frac{(b-a)^3}{24}\frac{1}{M^2}f''(\xi) = \frac{(3-1)^3}{24}\frac{f''(\xi)}{M^2} = \frac{1}{3}\frac{f''(\xi)}{M^2} \le 0.005$$

$$\Leftrightarrow \quad M^2 \ge \frac{f''(\xi)}{0.015} \underset{(M>0)}{\Leftrightarrow} M \ge \sqrt{\frac{f''(\xi)}{0.015}} \quad \Leftarrow \quad M \ge \sqrt{\frac{0.89617}{0.015}} = \sqrt{59.74} = 7.729 \quad \Leftarrow \quad \boxed{M=8}$$

These inequalities are trivially satisfied if $f''(\xi) \le 0$, so we only considered $f''(\xi) > 0$. ■

This is, of course, a priori. A posteriori fewer subintervals typically suffice. In this case, applying the compound Midpoint rule with $M=8$ subintervals the resulting error is 0.001914, well below the 0.005 tolerance, as expected. But the error with 7 subintervals is 0.002506; with 6 subintervals it is

0.003424; and with 5 it is 0.0049621, all of them still good enough. Only with 4 subintervals or less do we get an error greater than $\varepsilon = 0.005$, like 0.007851 for $M = 4$.

d)

Our subintegral function $f(x)$ looks like a Gauss-Hermite one, but ours is proper (between finite limits a, b), and there's no way around this. The only Gauss rules we know for proper integrals are Gauss-Legendre and Gauss-Chebyshev; but our $f(x)$ lacks a factor $w(x) = (1-x^2)^{-1/2}$ or anything similar to a Gauss-Chebyshev integral. One can artificially multiply and divide $f(x)$ by the Gauss-Chebyshev weight function[4], but Gauss-Legendre quadrature will probably work best.

So I will use 2 Gauss-Legendre nodes, then 3, then 4, etc., until the distance between the last two approximations of I obtained does not exceed the tolerance $\varepsilon = 0.005$. I will pick the nodes and weights from the table in the Appendix, and do the operations with Matlab. The task is particularly simple because the length of the interval [1,3] is the same as that of [−1,1], so one only needs to add 2 units to the nodes and leave the weights unchanged from the table:

```
>> f = @(x) exp(-x.^2).*(x.^2+1);              % vectorized f(x)
>> xi = [-1/sqrt(3) 1/sqrt(3)]+2;              % 2 NODES from table (n=1)
>> wi = [1 1];                                 % weights
>> Q2 = sum(wi.*f(xi))                         % Q for 2 nodes
Q2 =  0.40953
>> xi = [-0.77460 0 0.77460]+2;                % 3 NODES from table (n=2)
>> wi = [0.55556 0.88889 0.55556];             % weights from table
>> Q3 = sum(wi.*f(xi))                         % Q for 3 nodes
Q3 =   0.3932
>> termination = abs(Q3-Q2)                    % if > 0.005 will continue
termination =    0.016327
>> xi = [-0.86114 -0.33998 0.33998 0.86114]+2; % 4 NODES from table (n=3)
>> wi = [0.34785 0.65215 0.65215 0.34785];     % weights from table
>> Q4 = sum(wi.*f(xi))                         % Q for 4 nodes
Q4 =  0.39268
>> termination = abs(Q4-Q3)                    % if < 0.005 will terminate
termination =    0.00052442
```

Hence $\boxed{Q_{d)} = 0.39268}$ ■

Note that, theoretically, this procedure does not guarantee that the error made is less than 0.005 in the general case—just that the last two values obtained differ in less than that. Only when the error is halved at each iteration do both numbers coincide. However, in practice the convergence is typically faster than that, so the precision more than met. In our case not only does Q_4 comply (as it turns, $I - Q_4 = 0.00015$), but also Q_3 (turns out $I - Q_3 = -0.00037$). Not Q_2, though ($I - Q_2 = -0.0167$).

e)

The Gaussian rule "required" 4 evaluations of $f(x)$, while the Midpoint rule only 8. Gauss wins 4/8.

The computational cost is measured just as the total number of evaluations of $f(x)$ that must be performed (or the total number of nodes used) because the other operations are just sums and products that go very fast. Of course if $f(x)$ is algebraic, like $x^2/(x+2)$, then evaluating $f(x)$ is also very fast; but as soon as $f(x)$ contains some transcendental function(s) its evaluation is slower than sums and prod-

[4] This might even make some sense if you do not have the table of nodes and weights for the quadrature rule at hand, because the ones of the Gauss-Chebyshev quadrature are very easy to obtain.

ucts. Once $f_i = f(x_i)$ has been calculated, it just has to be multiplied by a number w_i (which is looked up in a table very fast) and added to the current value of Q (fast again). ■

Finally note that the 4-to-8 score considers our prior "sufficient" numbers of nodes. The posterior "necessary" numbers turned out to be 5 for the Midpoint rule and 3 for the Gauss-Legendre one. Gauss and Legendre beat Newton and Cotes again, this time 3 to 5.

Exercise 3

a)

Calling $u_1 = y_1$, $u_2 = y_1'$, $u_3 = y_2$, $u_4 = y_2'$:

$$\begin{cases} u_1' = u_2 \\ u_2' = \dfrac{-k_1 - k_2}{m_1} u_1 + \dfrac{k_2}{m_1} u_3 \\ u_3' = u_4 \\ u_4' = \dfrac{k_2}{m_2} u_1 + \dfrac{-k_2}{m_2} u_3 \end{cases} \quad \equiv \quad \mathbf{u}' = \underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 \\ \dfrac{-k_1 - k_2}{m_1} & 0 & \dfrac{k_2}{m_1} & 0 \\ 0 & 0 & 0 & 1 \\ \dfrac{k_2}{m_2} & 0 & \dfrac{-k_2}{m_2} & 0 \end{pmatrix}}_{J} \mathbf{u} = \mathbf{f}(t, \mathbf{u})$$

For $m_1 = m_2 = 1$, $k_1 = 1$, $k_2 = 2$:

$$\boxed{\mathbf{u}' = J\,\mathbf{u} = \mathbf{f}(t, \mathbf{u}) \quad \text{where} \quad J = \begin{pmatrix} 0 & 1 & 0 & 0 \\ -3 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & -2 & 0 \end{pmatrix}}$$ ■

b)

The initial conditions are: $t_0 = 0$; $\mathbf{u_0} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$

I will use step size $h = 1$ because it will take me to $t_1 = 1$ in just one step.

This is the advance formula of the RK4 method:

$$\begin{cases} \mathbf{k_1} = \mathbf{f}(t_k, \mathbf{u_k}) h_k \\ \mathbf{k_2} = \mathbf{f}(t_k + h_k/2, \mathbf{u_k} + \mathbf{k_1}/2) h_k \\ \mathbf{k_3} = \mathbf{f}(t_k + h_k/2, \mathbf{u_k} + \mathbf{k_2}/2) h_k \\ \mathbf{k_4} = \mathbf{f}(t_k + h_k, \mathbf{u_k} + \mathbf{k_3}) h_k \end{cases} \quad \rightarrow \quad \mathbf{u_{k+1}} = \mathbf{u_k} + \frac{\mathbf{k_1} + 2\mathbf{k_2} + 2\mathbf{k_3} + \mathbf{k_4}}{4}$$

With Matlab:

```
>> f = @(t,u) [0 1 0 0; -3 0 2 0; 0 0 0 1; 2 0 -2 0] * u;
>> h = 1; t0 = 0; t1 = h; u0 = [1 0 1 0]';
>> k1 = f(t0,u0)*h
k1 =
     0
    -1
     0
     0
>> k2 = f(t0+h/2,u0+k1/2)*h
k2 =
                      -0.5
                        -1
                         0
                         0
>> k3 = f(t0+h/2,u0+k2/2)*h
k3 =
                      -0.5
                     -0.25
                         0
                      -0.5
>> k4 = f(t0+h,u0+k3)*h
k4 =
                     -0.25
                       0.5
                      -0.5
                        -1
>> u1 = u0 + (k1+2*k2+2*k3+k4)/6
u1 =
                     0.625
                      -0.5
          0.916666666666667
         -0.333333333333333
```

Let us check that the result is good:

```
>> [tout,yout] = anm_ode(f,[0 1],u0,h,'RK4')
tout =
     0     1
yout =
                         1                     0.625
                         0                      -0.5
                         1          0.916666666666667
                         0         -0.333333333333333
```

The displacements y_1, y_2 are u_1, u_3 respectively, so:

$$\boxed{y_1(1) \approx 0.625\,; \quad y_2(1) \approx 0.916\hat{6}}$$

c)

We have to check that the method will be stable, which will happen iff the eigenvalues

eigs(hJ) = h eigs(J) are in the absolute stability region of the RK4 method:

$$\lambda^4 + 5\lambda^2 + 2 = 0 \quad \Rightarrow \quad \lambda^2 = \frac{-5 \pm \sqrt{25-8}}{2} = \begin{cases} (-5-\sqrt{17})/2 = -4.561553 & \Rightarrow \quad \begin{cases} \lambda_1 = 2.13578i \\ \lambda_2 = -2.13578i \end{cases} \\ (-5+\sqrt{17})/2 = -0.438447 & \Rightarrow \quad \begin{cases} \lambda_3 = 0.66215i \\ \lambda_4 = -0.66215i \end{cases} \end{cases}$$

Matlab will corroborate that these values are correct:

```
>> eigs([0 1 0 0; -3 0 2 0; 0 0 0 1; 2 0 -2 0])
ans =       0 -     2.1358i
            0 +     2.1358i
  7.7037e-034 -    0.66215i
  7.7037e-034 +    0.66215i
```

Since $h = 1$, these are the values that must be in the absolute stability region. Inspecting the figure provided shows that that is indeed the case (all λ_i are located on the vertical, imaginary axis). Therefore,

the step size $h = 1$ is adequate for RK4.

d)

Now $h = 0.5$ instead of 1, so eigs(hJ) are now half the men the used to be:

$$\lambda_1 = 1.0679i\,; \quad \lambda_2 = -1.0679i\,; \quad \lambda_3 = 0.33108i\,; \quad \lambda_4 = -0.33108i$$

Inspecting the figure of absolute stability regions shows that these points do not all belong to the RK1 region nor to the RK2 region, but they do belong to the other two regions. Hence,

the step size $h = 0.5$ is adequate for RK3 and RK4, but neither RK1 nor RK2.

Exercise 4

a)

To estimate f' we need at least two nodes (think secant); to estimate f'', three; and to estimate $f^{3)}$, four.

b)

The nodes are: $x_0 = z\,; \quad x_1 = z + h\,; \quad x_2 = z + 2h\,; \quad x_3 = z + 3h$

We will call $x_i = z + h_i$

Accordingly: $h_0 = 0\,; \quad h_1 = h\,; \quad h_2 = 2h\,; \quad h_3 = 3h$

If $f \in C^5([z,x_i])$, we can write the following Taylor series expansion (with Taylor remainder of order 5 because I would like to have $f^{4)}(z)$ in the error term):

$$f(x_i) = f(z) + f'(z)h_i + \frac{f''(z)}{2!}h_i^2 + \frac{f^{3)}(z)}{3!}h_i^3 + \frac{f^{4)}(z)}{4!}h_i^4 + \frac{f^{5)}(\xi_i)}{5!}h_i^5 \text{ for some } \xi_i \in (z,x_i)$$

Substituting into the numerical differentiation formula we are trying to determine:

$$f^{3)}(z) = A_0 f(x_0) + A_1 f(x_1) + A_2 f(x_2) + A_3 f(x_3) + E = \sum_{i=0}^{3} A_i f(x_i) + E =$$

$$= \sum_{i=0}^{3} A_i \left[f(z) + f'(z)h_i + \frac{f''(z)}{2!}h_i^2 + \frac{f^{3)}(z)}{3!}h_i^3 + \frac{f^{4)}(z)}{4!}h_i^4 + \frac{f^{5)}(\xi_i)}{5!}h_i^5 \right] + E =$$

$$= \underbrace{\left(\sum_{i=0}^{3} A_i\right)}_{=0} f(z) + \underbrace{\left(\sum_{i=0}^{3} A_i h_i\right)}_{=0} f'(z) + \underbrace{\left(\frac{1}{2!}\sum_{i=0}^{3} A_i h_i^2\right)}_{=0} f''(z) + \underbrace{\left(\frac{1}{3!}\sum_{i=0}^{3} A_i h_i^3\right)}_{=1} f^{3)}(z) +$$

$$+ \underbrace{\frac{1}{4!}\left(\sum_{i=0}^{3} A_i h_i^4\right) f^{4)}(z) + \frac{1}{5!}\left(\sum_{i=0}^{3} A_i h_i^5 f^{5)}(\xi_i)\right) + E}_{=0}$$

We must now solve the first four equations for A_i (and then the last one for E, if we want to obtain the error term). If we can do that we will have $f^{3)}(z)$ on both sides and all will be fine.

To obtain the coefficients:
$$\begin{cases} A_0 + A_1 + A_2 + A_3 = 0 \\ A_0 0 + A_1 h + A_2 2h + A_3 3h = 0 \\ A_0 0^2 + A_1 h^2 + A_2 (2h)^2 + A_3 (3h)^2 = 0 \\ A_0 0^3 + A_1 h^3 + A_2 (2h)^3 + A_3 (3h)^3 = 3! \end{cases}$$

This system can also be obtained by the method of indeterminate coefficients, i.e. by imposing the exact differentiation of 1, x, x^2 and x^3. A good way to solve these things "by hand" is to write the extended coefficient matrix and apply Gauss-like elementary row transformations:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 1 & 4 & 9 & 0 \\ 0 & 1 & 8 & 27 & 6/h^3 \end{pmatrix} \begin{matrix} \\ \langle F_3 - F_2 \rangle \\ \langle F_4 - F_2 \rangle \\ \\ \end{matrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 2 & 6 & 0 \\ 0 & 0 & 6 & 24 & 6/h^3 \end{pmatrix} \begin{matrix} \\ \\ \langle F_4 - 3F_3 \rangle \\ \langle F_3/2 \rangle \\ \\ \end{matrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 3 & 0 \\ 0 & 0 & 0 & 6 & 6/h^3 \end{pmatrix}$$

Backward substitution. From last equation: $A_3 = 1/h^3$
Substituting into the previous one: $A_2 = -3/h^3$
Substituting into the previous one: $A_1 = 3/h^3$
Substituting into the previous one: $A_0 = -1/h^3$

Therefore:
$$\boxed{f^{3)}(z) = \frac{-f(z) + 3f(z+h) - 3f(z+2h) + f(z+3h)}{h^3} + E}$$ ■

The error term has not been asked, but let us obtain it anyway:

$$E = \frac{-1}{4!}\left(\sum_{i=0}^{3} A_i h_i^4\right) f^{4)}(z) + \frac{-1}{5!}\left(\sum_{i=0}^{3} A_i h_i^5 f^{5)}(\xi_i)\right) =$$

$$= \frac{-1}{24}\left(0^4 + \frac{3}{h^3}h^4 + \frac{-3}{h^3}(2h)^4 + \frac{1}{h^3}(3h)^4\right) f^{4)}(z) + \frac{-1}{120}\left(0^5 + \frac{3}{h^3}h^5 f^{5)}(\xi_1) + \ldots\right) =$$

$$= \frac{-h}{24}(3 - 48 + 81) f^{4)}(z) + O(h^2) = \boxed{E = \frac{-3f^{4)}(z)}{2}h + O(h^2)}$$

This is consistent with the error term provided, $E = -3f^{4)}(\xi)\, h/2 + O(h^2)$ for some $\xi \in [z, z+3h]$ (think $\xi = z$), but our term is better because z is known, while ξ isn't. However, even if using $f(z)$ would have been slightly more convenient, to solve the exercise we will still use the term provided.

Furthermore, the term E we derived is also easier to obtain than by differentiating the error term $e(x)$ of the interpolation polynomial several times. And its derivation only relies on well-known Taylor series expansions, without making use of any niche theorem on how to differentiate divided differences. If, in spite of this, you are still interested in how to derive E by differentiating $e(x)$, please see the Appendix at the very end of this document.

c)

The optimal distance between nodes h_{opt} is the one minimizing the upper bound of the absolute value

of the total error E_{tot}, which is the truncation error E above (henceforth called E_t) plus the roundoff error E_r: $E_{tot} = f^{3)}(z) - \overline{D} = \underbrace{[f^{3)}(z) - D]}_{E=E_t} + \underbrace{[D - \overline{D}]}_{E_r} = E_t + E_r \quad \Rightarrow \quad |E_{tot}| \le |E_t| + |E_r|$

Here D is the result of the numerical differentiation formula with no perturbations and exact arithmetic, and $\overline{D}$ is the value actually obtained with real-world perturbations and finite-precision arithmetic.

To estimate h_{opt} we will completely neglect the term $O(h^2)$ in E, because the principal part of E_t is a $O(h)$[5] and we expect h_{opt} to be small. Then if M is an upper bound of $|f^{4)}(\xi)|$:

$$E_t = \frac{-3f^{4)}(\xi)}{2}h \quad \Rightarrow \quad |E_t| \le \frac{3M}{2}h \quad \text{(since } h > 0\text{)}$$

On the other hand, as seen in "the theory", an upper bound of $|E_r|$ is obtained as:

$$|E_r| \le AF \cdot \varepsilon$$

where ε is an upper bound of $|f_i - \overline{f}_i|$, i.e. of the absolute values of the errors in the nodal values of f, and the amplification factor AF is equal to the sum of the absolute values of the coefficients A_i:

$$AF = \sum_{i=0}^{3} |A_i| = \frac{1}{h^3} + \frac{3}{h^3} + \frac{3}{h^3} + \frac{1}{h^3} = \frac{8}{h^3}$$

Substituting: $$|E_{tot}| \le \frac{3M}{2}h + \frac{8}{h^3}\varepsilon = g(h)$$

To minimize this we solve $g'(h) = 0$ for h:

$$\frac{3M}{2} + 8\varepsilon(-3h^{-4}) = 0 \quad \Rightarrow \quad h^{-4} = \frac{3M/2}{24\varepsilon} = \frac{M}{16\varepsilon} \quad \Rightarrow \quad \boxed{h_{opt} = \sqrt[4]{\frac{16\varepsilon}{M}} = 2\sqrt[4]{\varepsilon/M}} \qquad \blacksquare$$

d)

We just need numeric values of ε, M to substitute above. Starting with the upper bound M of $|f^{4)}(\xi)|$:

$$f(x) = 2\sin(2x) \quad \Rightarrow \quad f^{4)}(\xi) = 32\sin(2\xi) \quad \text{with } \xi \in [z, z+3h]$$

The interval $[z, z+3h]$ that ξ belongs to will be very small, so M will be very close to $f^{4)}(z)$:

$$f^{4)}(z) = f^{4)}(\pi/12) = 32\sin(\pi/6) = 32/2 = 16$$

Even if $|f^{4)}|$ increases a tiny bit from z to $z+3h$, we still will use $\underline{M = 16}$, because we are just *estimating* h_{opt}. Of course one can always use $M = 32$, since $|\sin(2x)|$ will never exceed 1, and be formally correct; but our estimation of h_{opt} will probably be better with this tighter and almost correct value for M.

As for ε, we are instructed to use $\varepsilon = 10^{-16}$ to represent Matlab's default arithmetic. In the absence of other sources of noise, the value of ε depends on the precision of the arithmetic used. By default Matlab uses double-precision arithmetic, which means about 16 base-10 significant digits, so using $\varepsilon = 10^{-16}$ is indeed a reasonable approximate value. Substituting:

$$\sqrt[4]{\frac{16\varepsilon}{M}} = \sqrt[4]{\frac{16 \times 10^{-16}}{16}} = \boxed{h_{opt} = 1 \times 10^{-4}}$$

This means that steps sizes h on the order of 0.0001 should be close to minimizing the upper bound of $|E_{tot}|$ in the evaluation of $f^{3)}(\pi/12)$ using our formula in Matlab with its default arithmetic. That upper bound should be on the order of $g(10^{-4})$:

$$g(10^{-4}) = \frac{3M}{2}h + \frac{8}{h^3}\varepsilon = \frac{3 \times 16}{2}10^{-4} + \frac{8}{10^{-12}}10^{-16} = 32 \times 10^{-4} = \boxed{0.0032}$$

[5] Usually read "big O of h", and hence the indefinite article "a" instead of "an".

For those of you using $M = 32$, here are the corresponding results:

$$\sqrt[4]{\frac{16\times10^{-16}}{32}} = \underline{h_{opt} = 0.84\times10^{-4}}\,; \quad g(10^{-4}) = \frac{3\times32}{2}10^{-4} + \frac{8}{10^{-12}}10^{-16} = \underline{0.0054} \qquad \blacksquare$$

At first sight both $h_{opt} = $ 1e-4 and 0.0032 strike as too large. With double-precision arithmetic, i.e. about 16 orders of magnitude of precision at our disposal, it is surprising that we cannot use a few more of them to make h smaller than 1e-4 without compromising roundoff errors, or expect a maximum error significantly smaller than 0.0032. But numerical differentiation is an inherently unstable numerical process. In our case, with $h = $ 1e-4, the denominator of our differentiation formula is $h^3 = $ 1e-12, and its numerator is a difference of {values on the order of 1e0} that is on the order of 1e-12. We actually don't have so many "spare orders of magnitude" to use.

e)

Here are some of the assumptions made that are not satisfied exactly:

- I used $M = 16$, which is very slightly smaller than correct. You probably used $M = 32$, which is about twice as large as an ideal tight upper bound.

- We have neglected $O(h^2)$ in E. This is reasonable if h is small enough, because the principal part of E is a $O(h)$. It is even more reasonable in the context of an exam, in which we must follow instructions.

- The theory that shows AF to be equal to the sum of $|A_i|$ assumes that only the nodal ordinates $f(x_i)$ are perturbed ($\bar{f}_i$ instead of f_i), but not the abscissas (i.e. the nodes x_i) nor the operations carried out with all those values. If a system has finite-precision arithmetic, it will affect all of the above.

- The "16 significant digits" of double-precision arithmetic is only a rule of thumb, because digital computers operate in base 2, not base 10. This arithmetic does not use exactly 16 significant decimal digits, and ε is not exactly 10^{-16}. A better equivalent to the number of base-10 significant digits of double-precision numbers is $53\log_{10}(2) \approx 15.955$.

- More importantly, the distance between two "consecutive" real numbers that the computer can tell apart depends on their magnitude. For Matlab's double-precision arithmetic, the next real number after 1.0 is 1+eps(1), or about 1.0+2.22e-16; but after 1e16 comes 1e16+2, and after 1e20 comes 1e20+16384. In our case, $f(z) = 2\sin(2z) = 2\sin(\pi/6) = 2/2 = 1$, so Matlab's 2.22e-16 means the precision is about half the one corresponding to the value $\varepsilon = 10^{-16}$ we used.

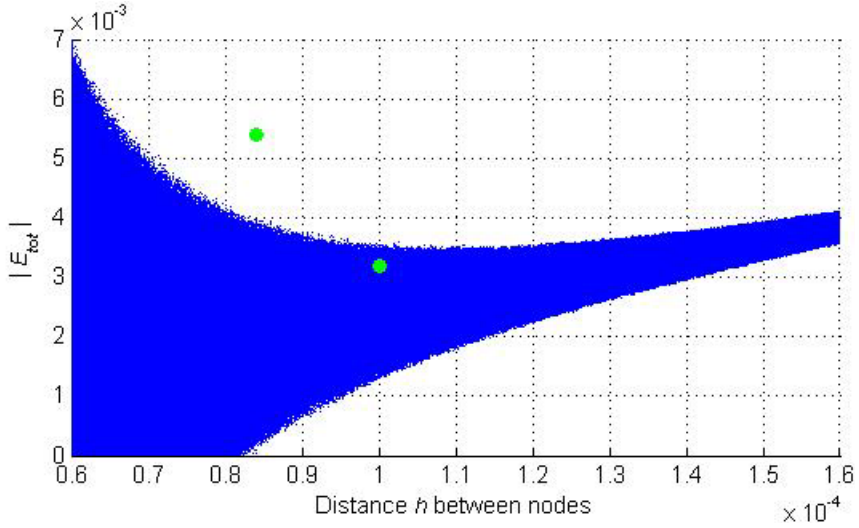
f)

First we must understand the figure well. Ten million random values of h around $h_{opt} = $ 1e-4 have been generated to produce ten million estimates of $f^{3)}(\pi/12)$ using the formula we obtained in b) with Matlab's default arithmetic. Subtracting them from the exact value (which is $-8\times3^{1/2}$) we get the corresponding errors. Each point in the figure represents one estimation, its abscissa being the value of h used, and its ordinate the absolute total error made.

The following code generates the figure again and adds two points, one for each one of our estimations (with $M = 16$ and with $M = 32$) of h_{opt} and of the corresponding maximum absolute error:

```
f = @(x) 2*sin(2*x);              % f(x)
f3exact = -16*cos(2*pi/12);       % its 3rd derivative at z=pi/12 (exact)
f3num = @(z,h) ((-f(z)+f(z+3*h))+3*(f(z+h)-f(z+2*h)))./h.^3;  % our formula
h1 = 0.6e-4;  h2 = 1.6e-4;        % min and max values of h for plot
hh = rand(1e7,1).^4*(h2-h1)+h1; % many random values of h between h1 and h2
f3numeric = f3num(pi/12,hh);      % numerically calculated values of f3(z)
ee = abs(f3exact - f3numeric);  % corresponding absolute errors
plot(hh,ee,'.','MarkerSize',1); axis([h1 h2 0 7e-3]); hold on   % blue dots
xlabel('Distance {\ith} between nodes');  ylabel('| {\itE_{tot}} |')
plot([1e-4 .84e-4],[.0032 .0054],'og','MarkerFace','g')    % our estimations
```

If you read the code carefully you should be able to understand all of it. (The `.^4` in the assignment of `hh` just pushes the random values a bit more to the left.)



The superior envelope curve of the cloud of points seems to have a minimum near h_{opt} = 1e-4 (it actually looks closer to 1.1e-4). The corresponding minimum absolute error upper bound seems to be about 3.5e-3, which is not too far from the predicted value 0.0032. Taking into account the number of assumptions accepted that are not really satisfied (see part e) above), the real behavior is surprisingly well predicted by the theory, and our predictions in section d) were good.
Even the point predicted with $M = 32$ is not too bad (although worse than the other one). ■

Some final comments follow. It can be seen in the figure that the roundoff errors grow much faster when h decreases than the truncation errors do when h increases. Picking a value of h that is too small is easy if you don't know well what you are doing, and can have "catastrophic" effects. Observe this:

```
>> f3exact = -16*cos(2*pi/12);
>> hh = [1e-4 1e-5 1e-6 1e-7];
>> f3numeric = f3num(pi/12,hh)      % numerically calculated f3
f3numeric =  -13.854      -14.322      -555.11  1.1102e+005
>> ee = abs(f3exact - f3numeric)    % absolute errors
ee =       0.0027105       0.46547        541.26  1.1104e+005
```

Also note that the error 0.0027105 for $h = h_{opt}$ is in no contradiction with $g(h_{opt})$ being 0.0032 because that is an expected upper bound of the error, but most of the times the error will be smaller.

And finally observe that the implementation of our numerical differentiation formula has grouped together the terms in the numerator in a way that tries to reduce roundoff errors:

```
f3num = @(z,h) ( (-f(z)+f(z+3*h)) + 3*(f(z+h)-f(z+2*h)) ) ./ h.^3;
```

In scientific computing, for better results, differences of very similar values should typically be calculated first (put into parentheses) and only later multiplied by the common factor 3 (in the second case). You can easily try applying the formula "as is":

```
f3num = @(z,h) ( -f(z) + 3*f(z+h) - 3*f(z+2*h) + f(z+3*h) ) ./ h.^3;
```

and re-generate the figure provided in the exercise statement. You will see that the results are visibly worse (although not catastrophically so).

Appendix: Long way to E[6]

Let's also see how to obtain the error term E of D by differentiating the error term $e(x)$ of the rule's interpolation polynomial. The error of the differentiation formula is the exact value $f^{3)}(z)$ minus its approximate value D. Since D is of interpolatory kind, in our case using 4 nodes, $D=p_3^{3)}(z)$, and:

$$E = f^{3)}(z) - D = f^{3)}(z) - p_3^{3)}(z) = \left[f(z) - p_3(z)\right]^{3)} = e^{3)}(z)$$

Hence the error of the approximation to the third derivative is the third derivative of the error of the approximation (i.e. interpolation) polynomial. Substituting the well-known expression of $e(z)$ in terms of a divided difference, and differentiating three times:

$$E = e^{3)}(z) = \left[f[x_0,x_1,x_2,x_3,z]\Pi(z)\right]^{3)} = \left[f[x_0,x_1,x_2,x_3,z]\Pi'(z) + f[x_0,x_1,x_2,x_3,z]'\Pi(z)\right]'' =$$

$$= \left[f[x_0,...,x_3,z]\Pi''(z) + f[x_0,...,x_3,z]'\Pi'(z) + f[x_0,...,x_3,z]'\Pi'(z) + f[x_0,...,x_3,z]''\Pi(z)\right]' =$$

$$= \left[f[x_0,...,x_3,z]\Pi''(z) + 2f[x_0,...,x_3,z]'\Pi'(z) + f[x_0,...,x_3,z]''\Pi(z)\right]' =$$

$$= f[x_0,...,x_3,z]\Pi'''(z) + f[x_0,...,x_3,z]'\Pi''(z) +$$

$$+2f[x_0,...,x_3,z]'\Pi''(z) + 2f[x_0,...,x_3,z]''\Pi'(z) +$$

$$+f[x_0,...,x_3,z]''\Pi'(z) + f[x_0,...,x_3,z]'''\Pi(z) =$$

$$= f[x_0,...,x_3,z]\Pi'''(z) + 3f[x_0,...,x_3,z]'\Pi''(z) + 3f[x_0,...,x_3,z]''\Pi'(z) + f[x_0,...,x_3,z]'''\Pi(z)$$

Now we will use the theorem: $f[x_0,...,x_n,z]^{k)} = k!\,f[x_0,...,x_n,\underbrace{z,...,z}_{k+1 \text{ times}}]$

and the well-known theorem: $f[x_0,...,x_n] = \dfrac{f^{n)}(\xi)}{n!}$ for some ξ between the nodes. Substituting:

$$E = f[x_0,...,x_3,z]\Pi'''(z) + 3\times 1!\,f[x_0,...,x_3,z,z]\Pi''(z) +$$

$$+3\times 2!\,f[x_0,...,x_3,z,z,z]\Pi'(z) + 3!\,f[x_0,...,x_3,z,z,z,z]\Pi(z) =$$

$$= \frac{f^{4)}(\xi_1)}{4!}\Pi'''(z) + 3\frac{f^{5)}(\xi_2)}{5!}\Pi''(z) + 6\frac{f^{6)}(\xi_3)}{6!}\Pi'(z) + 6\frac{f^{7)}(\xi_4)}{7!}\Pi(z) =$$

$$= \frac{f^{4)}(\xi_1)}{24}\Pi'''(z) + \frac{f^{5)}(\xi_2)}{40}\Pi''(z) + \frac{f^{6)}(\xi_3)}{120}\Pi'(z) + \frac{f^{7)}(\xi_4)}{840}\Pi(z) \qquad (1)$$

for some points ξ_1, ξ_2, ξ_3, ξ_4 between the nodes and z (or, in our case, simply between the nodes, since z is the first one).

On the other hand, taking into account that $x_0=z$, $x_1=z+h$, $x_2=z+2h$, $x_3=z+3h$:

$$\Pi(x) = (x-z)\,(x-z-h)\,(x-z-2h)\,(x-z-3h) \qquad (2)$$

The last term in (1) is 0 because the first term of (2) vanishes. If Matlab's symbolic toolbox is installed, it will agree with us:

---

[6] If this reminds you of a song by AC/DC, then there's something you're not pronouncing properly.

```
>> syms x z h                          % declare x, z, h as symbolic variables
>> Pi = (x-z)*(x-z-h)*(x-z-2*h)*(x-z-3*h)       % Pi(x)
Pi =   -(x - z)*(h - x + z)*(2*h - x + z)*(3*h - x + z)
>> Piz = subs(Pi,x,z)                           % Pi(z)
Piz =  0
```

For the other terms in (1) we will have to differentiate. I will assume we can all {differentiate polynomials} and let Matlab do the tedious work:

```
>> Pi1 = diff(Pi,x)             % 1st derivative of Pi(x) with respect to x
Pi1 =
(x - z)*(2*h - x + z)*(3*h - x + z) - (h - x + z)*(2*h - x + z)*(3*h - x +
z) + (x - z)*(h - x + z)*(2*h - x + z) + (x - z)*(h - x + z)*(3*h - x + z)
>> Pi1 = simplify(Pi1)          % simplified
Pi1 =  -2*(3*h - 2*x + 2*z)*(h^2 - 3*h*x + 3*h*z + x^2 - 2*x*z + z^2)
>> Pi1z = subs(Pi1,x,z)         % Pi1(z)
Pi1z =  -6*h^3
>> Pi2 = simplify(diff(Pi1,x))  % 2nd derivative with respect to x
Pi2 =  22*h^2 - 36*h*x + 36*h*z + 12*x^2 - 24*x*z + 12*z^2
>> Pi2z = subs(Pi2,x,z)         % Pi2(z)
Pi2z =  22*h^2
>> Pi3 = simplify(diff(Pi2,x))  % 3rd derivative with respect to x
Pi3 =  24*x - 36*h - 24*z
>> Pi3z = subs(Pi3,x,z)         % Pi3(z)
Pi3z =  -36*h
```

Substituting these results:

$$E = \frac{f^{4)}(\xi_1)}{24}(-36h) + \underbrace{\frac{f^{5)}(\xi_2)}{40}(22h^2)}_{O(h^2)} + \underbrace{\frac{f^{6)}(\xi_3)}{120}(-6h^3)}_{O(h^3)} + 0 = \frac{-3f^{4)}(\xi_1)}{2}h + O(h^2)$$

for some ξ_1 between the nodes, as the exercise statement asserted calling ξ instead of ξ_1. The first term marked as $O(h^2)$ does indeed tend to zero at least as fast[7] as h^2 (although it can tend faster if $f^{5)}(z) = 0$ with $f^{5)}$ continuous, because then ξ_5 will tend to z from the left). And of course the term marked $O(h^3)$ does indeed tend to zero at least as fast as h^3, so faster than like h^2, and can be absorbed into $O(h^2)$.

Finally consider how much easier it was to obtain E using Taylor series expansions than this (specially if you do Matlab's tedious differentiations and simplifications manually).

[7] We say that $f(h)$ tends to zero "at least as fast as" h^n (where $n \in \mathbb{N}$) if there exist two real constants k, ε (the latter often but not necessarily small) such that $\forall h$ with $|h| \le \varepsilon$, $|f(h)| \le k\,h^n$. We then write $f(h) = O(h^n)$ and say that $f(h)$ is "a big O of h to the n", or a "remainder of order n of h". Note that if $m > n$, $O(h^m)$ also tends to zero "at least as fast" as h^n (faster, actually) so, by definition, any $O(h^m)$ is also a $O(h^n)$ (but not vice-versa). This is a convenient definition; for instance, $O(h^m) + O(h^n) = O(h^n)$, and that's why we wrote $O(h^2) + O(h^3)$ above just as $O(h^2)$.