Numerical Analysis – Lab 8

1 Finite difference approximation of derivatives

Hydraulics[1]**.** The height $q(t)$ reached at time t by a fluid in a straight cylinder of radius $R = 1$ m with a circular hole of radius $r = 0.1$ m on the bottom, has been measured every 5 seconds yielding the following values

t	0	5	10	15	20
$q(t)$	0.6350	0.5336	0.4410	0.3572	0.2822

We want to compute an approximation of the emptying velocity $q'(t)$ of the cylinder, then compare it with the one predicted by Torricelli's law: $q'(t) = -\gamma(r/R)^2\sqrt{2gq(t)}$, where $g = 9.8$ m/s^2 is the modulus of gravity acceleration and $\gamma = 0.6$ is a correction factor. Let us try the 2-point forward/backward difference formulae and the 3-point forward/centered/backward difference formulae.

```
% derivative in hydraulics
t = 0:5:20; h = 5;
q = [0.6350 0.5336 0.4410 0.3572 0.2822];

% 2-point forward/backward differences
% forward for t=0:5:15; backwad t=5:5:20
dq2 = (q(2:end)-q(1:end-1))/h;
dq2f = [dq2 NaN]; dq2b = [NaN dq2];

% 3-point forward/center/backward differences
% forward for t=0:5:10
dq3f=(-3*q(1:end-2)+4*q(2:end-1)-q(3:end))/2/h;
dq3f=[dq3f NaN NaN];
% backward for t=10:5:20
dq3b=(-q(1:end-2)+4*q(2:end-1)-3*q(3:end))/2/h;
dq3b=[NaN NaN dq3b];
% centered for t=5:5:15
dq3c=(q(3:end)-q(1:end-2))/2/h;
dq3c=[NaN dq3c NaN];

% Torricelli's law
dq=-0.6*(0.1/1)^2*sqrt(2*9.80665*q);

% print the results
fprintf('\n2f  ');
disp(num2str(dq2f,'%.6f  '));
fprintf('\n2b  ');disp(dq2b);
fprintf('\n3f  ');disp(dq3f);
fprintf('\n3c  ');disp(dq3c);
fprintf('\n3b  ');disp(dq3b);
fprintf('\nq'' ');disp(dq);
fprintf('\n')
```

t	0	5	10	15	20
2f	-0.020280	-0.018520	-0.016760	-0.015000	NaN
2b	NaN	-0.020280	-0.018520	-0.016760	-0.015000
3f	-0.021160	-0.019400	-0.017640	NaN	NaN
3c	NaN	-0.019400	-0.017640	-0.015880	NaN
3b	NaN	NaN	-0.017640	-0.015880	-0.014120
$q'(t)$	-0.021175	-0.019410	-0.017646	-0.015881	-0.014116

In this example, the step size $h = 5$ is used for approximation of the derivative values at the given points. The error for the 2-point formulae is $\frac{h}{2}q''(\xi)$ for some ξ between the two points used. The errors for the 3-point endpoint (forward/backward) formulae are in the form $\frac{h^2}{3}q'''(\xi)$ for some ξ between the three points used, while the error for the 3-point midpoint (centered) formula is $\frac{h^2}{6}q'''(\xi)$ for some ξ between the three points used.

In practice, however, often the function q and the derivatives of q are unknown, so the errors can not be estimated with the error formulae. Then, it would be useful to try smaller step size h, and see whether the results are converging. If the results look converging, then practically one regards the converged results as accurate, which is reasonable from the error forms.

Akin to the h-refinement convergence test, one may increase the order of approximation (p-refinement), or equivalently number of points in the finite difference formulae. Let us try the 5-point formulae:

[1]The example is taken from the book *Scientific Computing with MATLAB and OCTAVE*.

$$f'(t) = \frac{1}{12h}\left[-25f(t) + 48f(t+h) - 36f(t+2h) + 16f(t+3h) - 3f(t+4h)\right] + \frac{h^4}{5}f^{(5)}(\xi),$$

$$f'(t) = \frac{1}{12h}\left[f(t-2h) - 8f(t-h) + 8f(t+h) - f(t+2h)\right] + \frac{h^4}{30}f^{(5)}(\xi),$$

which are the 5-point endpoint/midpoint formulae that we have learned. But since in the example there are only 5 data points, we need also the 5-point 2nd-point formula (**optional**) derived as follows

$$\begin{aligned} f(x) = {} & f(t-h)\frac{(x-t)(x-t-h)(x-t-2h)(x-t-3h)}{-h(-2h)(-3h)(-4h)} + \\ & f(t)\frac{(x-t+h)(x-t-h)(x-t-2h)(x-t-3h)}{h(-h)(-2h)(-3h)} + \\ & f(t+h)\frac{(x-t+h)(x-t)(x-t-2h)(x-t-3h)}{2h(h)(-h)(-2h)} + \\ & f(t+2h)\frac{(x-t+h)(x-t)(x-t-h)(x-t-3h)}{3h(2h)(h)(-h)} + \\ & f(t+3h)\frac{(x-t+h)(x-t)(x-t-h)(x-t-2h)}{4h(3h)(2h)(h)} + \\ & \frac{f^{(5)}(\xi(x))}{5!}(x-t+h)(x-t)(x-t-h)(x-t-2h)(x-t-3h), \end{aligned}$$

so

$$\begin{aligned} f'(t) = {} & f(t-h)\frac{1}{-4h} + f(t)\left(\frac{1}{h} + \frac{1}{-h} + \frac{1}{-2h} + \frac{1}{-3h}\right) + f(t+h)\frac{6}{4h} + \\ & f(t+2h)\frac{3}{-6h} + f(t+3h)\frac{2}{24h} - \frac{f^{(5)}(\xi(t))}{5!}6h^4 \\ = {} & \frac{1}{12h}\left[-3f(t-h) - 10f(t) + 18f(t+h) - 6f(t+2h) + f(t+3h)\right] - \frac{h^4}{20}f^{(5)}(\xi(t)). \end{aligned}$$

Combining all the 5-point formulae, we obtain for $t_i = t_0 + ih$, $i = 0, 1, \ldots, 4$,

$$\begin{pmatrix} f'(t_0) \\ f'(t_1) \\ f'(t_2) \\ f'(t_3) \\ f'(t_4) \end{pmatrix} = \frac{1}{12h}\begin{pmatrix} -25 & 48 & -36 & 16 & -3 \\ -3 & -10 & 18 & -6 & 1 \\ 1 & -8 & 0 & 8 & -1 \\ -1 & 6 & -18 & 10 & 3 \\ 3 & -16 & 36 & -48 & 25 \end{pmatrix}\begin{pmatrix} f(t_0) \\ f(t_1) \\ f(t_2) \\ f(t_3) \\ f(t_4) \end{pmatrix} + \frac{h^4}{60}\begin{pmatrix} 12f^{(5)}(\xi_0) \\ -3f^{(5)}(\xi_1) \\ 2f^{(5)}(\xi_2) \\ -3f^{(5)}(\xi_3) \\ 12f^{(5)}(\xi_4) \end{pmatrix},$$

where the matrix (including the factor $\frac{1}{12h}$) is called a differentiation matrix. We get

```
D = 1/12/h*[-25 48 -36 16 -3; -3 -10 18 -6 1;
    1 -8 0 8 -1; -1 6 -18 10 3; 3 -16 36 -48 25];
dq5 = D*q(:); dq5 = dq5';
fprintf('5  '); disp(num2str(dq5,'%.6f  ')); fprintf('\n');
fprintf('q''  '); disp(num2str(dq,'%.6f  ')); fprintf('\n');
```

t	0	5	10	15	20
5-point	-0.021160	-0.019400	-0.017640	-0.015880	-0.014120
q'	-0.021175	-0.019410	-0.017646	-0.015881	-0.014116

Compared with the 2-point and 3-point difference approximations, we find the 5-point approximations are the same as the 3-point approximations. So it is likely that the 3-point and 5-point results are accurate, but the accuracy is limited by the accuracy of the given data. A rigorous argument would require to compare the error formulae of 3-point and 5-point approximations, which then relies on the knowledge of the function and its derivatives. Practically, it is therefore recommended to do the h-refinement for convergence test, or the p-refinement convergence test should be done with a sufficiently small h.

Finite difference on general grids (optional)[2]**.** The difference formulae for approximation of the first derivative can also be derived on a general set of $n+1$ distinct points x_0, x_1, …, x_n. Recall the Lagrange interpolating polynomial of f is

$$\sum_{j=0}^{n} f(x_j)L_j(x) \quad \text{with } L_j(x) = \frac{1}{a_j}\prod_{\substack{k=0\\k\neq j}}^{n}(x-x_k), \quad a_j = \prod_{\substack{k=0\\k\neq j}}^{n}(x_j-x_k).$$

We have the approximation

$$f(x) = \sum_{j=0}^{n} f(x_j)L_j(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}\prod_{j=0}^{n}(x-x_j),$$

$$f'(x_i) = \sum_{j=0}^{n} f(x_j)L_j'(x_i) + \frac{f^{(n+1)}(\xi(x_i))}{(n+1)!}a_i,$$

where

$$L_j'(x_i) = \frac{1}{a_j}\prod_{\substack{k=0\\k\neq i,j}}^{n}(x_i-x_k) = \frac{a_i}{a_j(x_i-x_j)}, \quad \text{for } i\neq j,$$

$$L_i'(x_i) = \sum_{\substack{j=0\\j\neq i}}^{n}\frac{1}{x_i-x_j} = -\sum_{\substack{j=0\\j\neq i}}^{n} L_j'(x_i).^{3}$$

Therefore, in term of the differentiation matrix, all the $n+1$-point formulae are described as follows

$$\begin{pmatrix} f'(x_0)\\ f'(x_1)\\ \vdots\\ f'(x_{n-1})\\ f'(x_n)\end{pmatrix} = \begin{pmatrix} L_0'(x_0) & L_1'(x_0) & \dots & L_{n-1}'(x_0) & L_n'(x_0)\\ L_0'(x_1) & L_1'(x_1) & \dots & \vdots & \vdots\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ L_0'(x_n) & L_1'(x_n) & \dots & \dots & L_n'(x_n)\end{pmatrix}\begin{pmatrix} f(x_0)\\ f(x_1)\\ \vdots\\ f(x_{n-1})\\ f(x_n)\end{pmatrix} + \frac{1}{(n+1)!}\begin{pmatrix} f^{(n+1)}(\xi_0)a_0\\ f^{(n+1)}(\xi_1)a_1\\ \vdots\\ f^{(n+1)}(\xi_{n-1})a_{n-1}\\ f^{(n+1)}(\xi_n)a_n\end{pmatrix}.$$

We then code a function to calculate the differentiation matrix.

```
function D = dmat(x)
n = numel(x)-1;
x = x(:);
D = x - x'; D = D + eye(n+1);
a = prod(D,2);
D = (a*(1./a'))./D;
for i=1:n+1
    D(i,i) = 0;
    D(i,i) = - sum(D(i,:));
end % function file ends
```

```
f=@(x) sin(pi*x); fp=@(x) pi*cos(pi*x);
x = linspace(-1,1,11)';
D = dmat(x); Df = D*f(x);
fprintf('Df  ');
disp(num2str(Df','%.6f  '));
fprintf('\nf''');
disp(num2str(fp(x)','%.6f  '));
fprintf('\nerr');
disp(num2str(fp(x)'-Df','%g  '));
fprintf('\n');
```

x	-1.0000	-0.8000	-0.6000	-0.4000	-0.2000	0
11-point	-3.139359	-2.541830	-0.970754	0.970786	2.541613	3.141583
f'	-3.141593	-2.541602	-0.970806	0.970806	2.541602	3.141593
err	-0.00223351	0.000228553	-5.17163e-05	1.96477e-05	-1.13157e-05	9.45449e-06

x	0.2000	0.4000	0.6000	0.8000	1.0000
11-point	2.541613	0.970786	-0.970754	-2.541830	-3.139359
f'	2.541602	0.970806	-0.970806	-2.541602	-3.141593
err	-1.13157e-05	1.96477e-05	-5.17163e-05	0.000228553	-0.00223351

[2]The material is taken from *Spectral Methods in MATLAB*

[3]because when $f(x)\equiv 1$ we have the polynomial equal to 1 i.e. $\sum_{j=0}^{n} f(x_j)L_j(x) = \sum_{j=0}^{n} L_j(x) \equiv 1$.

We can see that the accuracy is less near the endpoints than in the middle. Now let us try with Runge's example. We just need to replace the definitions of f and f' in the above program with

```
f = @(x) 1./(1+25*x.^2);  fp = @(x) -50*x./(1+25*x.^2).^2;
```

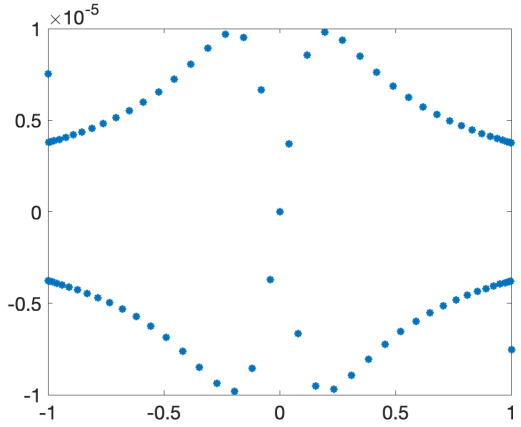
x	-1.0000	-0.8000	-0.6000	-0.4000	-0.2000	0
11-point	79.015837	-9.520362	3.036652	-0.568326	3.477376	-0.000000
f'	0.073964	0.138408	0.300000	0.800000	2.500000	0.000000
err	-78.9419	9.65877	-2.73665	1.36833	-0.977376	8.88693e-16

x	0.2000	0.4000	0.6000	0.8000	1.0000
11-point	-3.477376	0.568326	-3.036652	9.520362	-79.015837
f'	-2.500000	-0.800000	-0.300000	-0.138408	-0.073964
err	0.977376	-1.36833	2.73665	-9.65877	78.9419

Actually, the results would diverge near the endpoints if we increase the number of nodes. To circumvent the issue, we can use the Chebyshev nodes instead. Just replace the definition of `x` with

```
x = linspace(0,1,11)'; x = cos(pi*x); x = flip(x);
```

The error with $n = 80$ is shown below (note the scale of the y-axis).



For the Chebyshev nodes on $[-1, 1]$,

$$x_j = \cos\frac{(n-j)\pi}{n\pi}, \quad j = 0, 1, \ldots, n,$$

the general differentiation matrix can be simplified with

$$L_0'(x_0) = -\frac{2n^2+1}{6}, \quad L_n'(x_n) = \frac{2n^2+1}{6}, \quad L_j'(x_i) = \frac{c_i}{c_j}\frac{(-1)^{i+j}}{(x_i - x_j)}, \ i \neq j, \quad i, j = 1, \ldots, n-1$$

where

$$c_i = \begin{cases} 2, & i = 0 \text{ or } n, \\ 1, & \text{otherwise.} \end{cases}$$

For a general interval $[a, b]$, the Chebyshev nodes become

$$x_j = \frac{1}{2}(a + b + t_j(b - a)), \quad j = 0, \ldots, n$$

where t_j are the Chebyshev nodes on $[-1, 1]$, and the differentiation matrix becomes

$$D_{[a,b]} = \frac{2}{b-a} D_{[-1,1]}$$

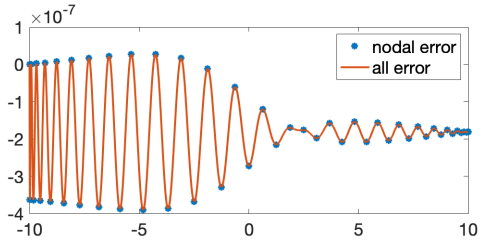
obtained by the change of variable $x = \frac{1}{2}(a + b + t(b - a))$ for $t \in [-1, 1]$.

With the differentiation matrix, not only the derivative can be evaluated numerically but also the antiderivative. To do that, we may want to invert the differentiation matrix with the derivative values as the right hand side vector, to get the function values on the nodes. This is straightforward from the equation of the differentiation matrix shown before. However, since the row sum of the differentiation matrix is zero, the matrix is singular, which is reasonable because the antiderivative is not unique (but unique up to constants). To fix the constant, we can specify the value of the antiderivative at one point. That is, we solve the differential equation

$$F'(x) = f(x), \quad F(x_0) = C.$$

The initial value condition $F(x_0) = C$ can be used to replace the condition $F'(x_0) = f(x_0)$, which changes the first row of the matrix and the first entry of the right hand side. Below is an example based on the Chebyshev differentiation matrix. The p.d.f. of the standard normal distribution is $f(x) = \mathrm{e}^{-x^2/2}/\sqrt{2\pi}$. Its c.d.f. is an antiderivative $F(x) = \int_{-\infty}^{x} f(t)\,\mathrm{d}t$, which is not an elementary function. Therefore, people defined a special function (called error function) $\mathrm{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-t^2}\,\mathrm{d}t$ to express the c.d.f. as $F(x) = (1 + \mathrm{erf}(x/\sqrt{2}))/2$. We evaluate F by our method and compare to using the built-in `erf`. The initial condition is an approximate one $F(-10) \approx 0$.

```
function [D,x]=chebdmat(a,b,n)
% assume a<b
t = linspace(0,1,n+1);
t = flip(cos(pi*t')); j = 0:n;
D = (t - t') + eye(n+1);
D = (-1).^(j'+j)./D;
D(2:n,[1,n+1]) = D(2:n,[1,n+1])/2;
D([1,n+1],2:n) = D([1,n+1],2:n)*2;
D(1,1) = -(2*n^2+1)/6;
D(n+1,n+1) = (2*n^2+1)/6;
for j=2:n
    %D(j,j) = -t(j)/2/(1-t(j)^2);
    D(j,j) = 0; D(j,j) = -sum(D(j,:));
end
x = (a+b + t*(b-a))/2;
D = 2/(b-a)*D; % function file ends
```




```
f = @(x) exp(-x.^2/2)/sqrt(2*pi);
F = @(x) (1+erf(x/sqrt(2)))/2;
[D,x] = chebdmat(-10,10,50);
D(1,:)=0; D(1,1)=1; y=f(x); y(1)=0; Y=D\y;
plot(x,F(x)-Y,'*','linewidth',2);
set(gca,'fontsize',18);xx=linspace(-10,10,1e4);
YY = lagrange(x,Y,xx);
plot(xx,F(xx)-YY,'linewidth',2);
```

Here, after having the nodal values of $F(x)$, we also evaluated the interpolating polynomial of $F(x)$.

Complex variable method (optional). Suppose $f : \mathbb{C} \to \mathbb{C}$ is an analytic function in the complex plane, and $f(\mathbb{R}) \subseteq \mathbb{R}$. An alternative idea for approximating the derivative $f'(x) \approx (f(x+h) - f(x))/h$ ($x \in \mathbb{R}$) is using *complex step differentiation*

$$f'(x) \approx \frac{\mathrm{Im}\, f(x + \mathrm{i}\, h)}{h}, \qquad h \in \mathbb{R} \text{ small}.$$

The error of the complex step formula can be found as follows:

$$f(x + \mathrm{i}\, h) = f(x) + f'(x)\mathrm{i}\, h + \frac{f''(x)}{2!}(\mathrm{i}\, h)^2 + \frac{f'''(\xi)}{3!}(\mathrm{i}\, h)^3,$$

taking the imaginary parts of both sides results

$$\mathrm{Im} f(x + \mathrm{i}\, h) = f'(x)h - \frac{\mathrm{Re} f'''(\xi)}{3!}h^3 \Longrightarrow f'(x) = \frac{\mathrm{Im}\, f(x + \mathrm{i}\, h)}{h} + \frac{\mathrm{Re} f'''(\xi)}{3!}h^2.$$

In addition to the higher order than the real step 2-point method, the complex step method is significantly less affected by round-off errors since it is free of cancellation. We compare the two formulas in approximating $f'(2)$ for

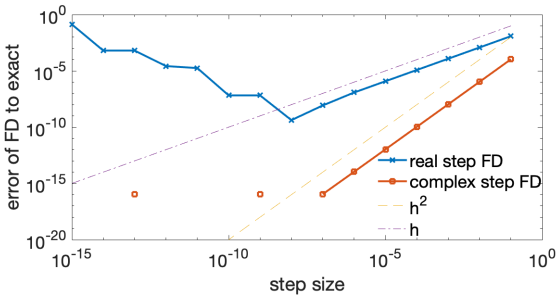
$$f(x) = \log(1 + (x-4)^2).$$

Plot the error of the two finite difference approximations against the step size $h \to 0$. In the following figure, the error of the complex step method quickly decays to the machine epsilon whereas that of the real step can increase when the step size is too small and the round-off error dominates.

```
f = @(x) log(1+(x-4)^2);
Df = @(x) 1/(1+(x-4)^2)*2*(x-4);
x = 2; Dfp = Df(x);

% real step finite difference
t = 10.^(-1:-1:-15);
rFDp = zeros(size(t));
for m = 1:length(t)
    rFDp(m) = (f(x+t(m))-f(x))/t(m);
end

% complex step finite difference
cFDp = zeros(size(t));
for m = 1:length(t)
    cFDp(m) = imag(f(x+1i*t(m)))/t(m);
end
```

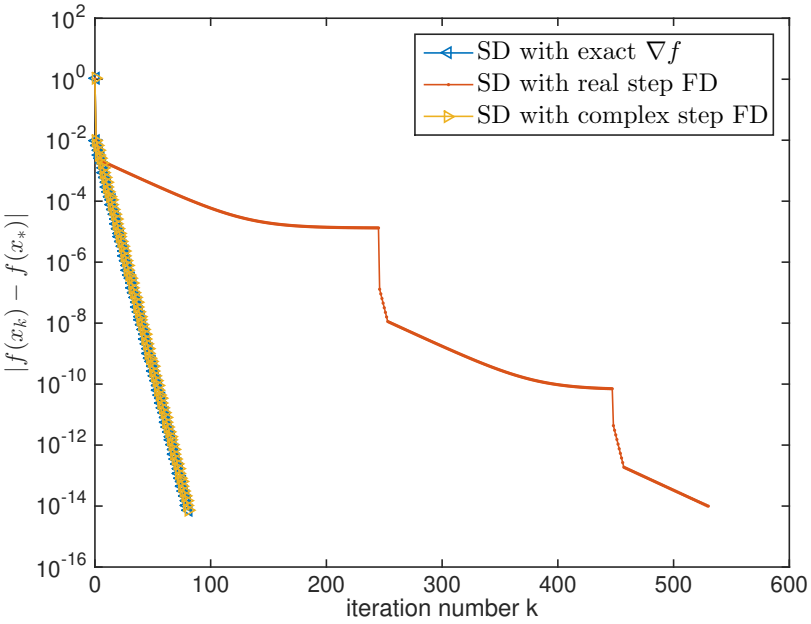



```
loglog(t,abs(rFDp-Dfp),'x-','linewidth',2);
hold on;
loglog(t,abs(cFDp-Dfp),'s-','linewidth',2);
loglog(t,t.^2,'--'); loglog(t,t,'-.'); hold off;
xlabel('step size');
ylabel('error of FD to exact');
```

You might wonder: is such a high accuracy of numerical derivative necessary? Let us try to use the numerical derivative for the steepest descent method with a line search (the step size, also known as the learning rate, is picked by optimizing along the gradient direction). We consider minimizing the following function[4]

$$f(x_1, x_2) = 10^{-2}x_1^2 + x_2^2.$$

The programs used are too complicated to be shown here; see `lab8ex5.m`, `SteepestDescent`, `linesch_ww`[5]. The results of using different gradients are shown in the following figure. We see that the accuracy of the gradient has a huge impact on the performance of the steepest descent method. The complex step differentiation gives almost the same performance as the closed-form differentiation does, whereas the real step differentiation slows down a lot the convergence.



[4]The example here has a closed-form gradient and also a closed form minimizer $(0, 0)$. So usually one would use the closed form. But it is often in application that a closed-form function is not known.

[5]The line search was downloaded from the web written by M. Overton. The other programs are home made.

Noisy data smoothing and differentiation (optional). Due to the time limit, this topic will be left for self-exploration. Search on the web for *Practical Aspects of the Equation-Error Method for Aircraft Parameter Estimation*, also Savitzky-Golay filter.

2 Richardson extrapolation (optional)

Although derived in different way, the Richardson extrapolation is effectively the same as Neville's method for extrapolation; see Lab 5. The extrapolated numerical derivative can get somewhat higher accuracy than the original numerical derivatives, but is still limited by the round-off errors. The Richardson extrapolation is based on the Taylor series of the error

$$M = N_1(h) + K_1 h + K_2 h^2 + k_3 h^3 + \dots$$

so that

$$2M = 2N_1(\frac{h}{2}) + K_1 h + 2K_2(\frac{h}{2})^2 + 2K_3(\frac{h}{2})^3 + \dots$$

$$\Longrightarrow M = 2N_1(\frac{h}{2}) - N_1(h) + K_2(-\frac{h^2}{2}) + K_3(-\frac{3h^2}{4}) + \dots$$

Thus $N_2(h) := 2N_1(\frac{h}{2}) - N_1(h)$ has the order higher by one than $N_1(h)$. Again, starting from

$$M = N_2(h) + \tilde{K}_2 h^2 + \tilde{K}_3 h^3 + \dots$$

we can get

$$4M = 4N_2(\frac{h}{2}) + \tilde{K}_2 h^2 + 4\tilde{K}_3(\frac{h}{2})^3 + \dots$$

$$\Longrightarrow 3M = 4N_2(\frac{h}{2}) - N_2(h) + \tilde{K}_3(-\frac{h^3}{2}) + \dots$$

Thus $N_3(h) := \frac{4}{3}N_2(\frac{h}{2}) - \frac{1}{3}N_2(h)$ has the order of error $O(h^3)$. We can verify that it corresponds to Neville's table for evaluating at $x = 0$:

$$\begin{array}{lll} h & N_1(h) & \\ \frac{h}{2} & N_1(\frac{h}{2}) & \frac{1}{(h/2)-h}\left((0-h)N_1(\frac{h}{2}) - (0-\frac{h}{2})N_1(h)\right) = N_2(h) \\ \frac{h}{4} & N_1(\frac{h}{4}) & \frac{1}{(h/4)-(h/2)}\left((0-\frac{h}{2})N_1(\frac{h}{4}) - (0-\frac{h}{4})N_1(\frac{h}{2})\right) = N_2(\frac{h}{2}) \quad N_3(h) \end{array}$$

We have used Neville's method for the extrapolation of the 2-point differences in Lab 5. Now we can try also the 3-point end-point and mid-point formulae. To that end, we need the Taylor series of the error. Suppose

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f'''(x_0)}{3!}(x - x_0)^3 + \dots$$

Taking $x = x_0 + h, x_0 + 2h$, respectively gives

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{f''(x_0)}{2!}h^2 + \frac{f'''(x_0)}{3!}h^3 + \frac{f^{(4)}(x_0)}{4!}h^4 + \dots,$$

$$f(x_0 + 2h) = f(x_0) + f'(x_0)2h + \frac{f''(x_0)}{2!}4h^2 + \frac{f'''(x_0)}{3!}8h^3 + \frac{f^{(4)}(x_0)}{4!}16h^4 + \dots$$

Our goal is to cancel the term with h^2 and recover the end-point formula. We get

$$f'(x_0) = \frac{1}{2h}\left[-3f(x_0) + 2f(x_0 + h) - f(x_0 + 2h)\right] - 4\frac{f'''(x_0)}{3!}h^2 - 12\frac{f^{(4)}(x_0)}{4!}h^3 + \dots$$

Thus, we expect the Richardson extrapolation works but with $K_1 = 0$, and each time extrapolation raises the order by one. Since $K_1 = 0$, one can also simply set $N_2 = N_1$ without affecting the order of errors even though the result is different from calculating N_2 as normal.

The situation is quite different for the mid-point formula. Let $x = x_0 - h$ in the Taylor series of f. We obtain

$$f(x_0 - h) = f(x_0) - f'(x_0)h + \frac{f''(x_0)}{2!}h^2 - \frac{f'''(x_0)}{3!}h^3 + \frac{f^{(4)}(x_0)}{4!}h^4 + \dots.$$

Subtracting $f(x_0 + h)$ with $f(x_0 - h)$ does not only cancel the term with h^2 but also all the even powers of h, which leads to

$$f'(x_0) = \frac{1}{2h}\left[f(x_0 + h) - f(x_0 - h)\right] + \frac{f'''(x_0)}{3!}h^2 + \frac{f^{(5)}(x_0)}{5!}h^4 + \dots$$

We can think of the expansion in term of $\tilde{h} = h^2$ and then it falls back to the previous situation. So Neveill's table is calculated for the nodes $\tilde{h}$, $\tilde{h}/4$, $\tilde{h}/16$, ... in lieu of h, $h/2$, $h/4$, ... Now we can do the experiments.

```
f = @sin; h = 2.^(-1:-1:-9); x = 1;
y = f(x); y1 = f(x+h); y2 = f(x+2*h); yn = f(x-h);

% 3-point forward differences
fpfw = (-3*y+4*y1-y2)/2./h; [fpfwe, Qfw] = neville(h,fpfw,0);

% centered differences
fpc = (y1-yn)/2./h; [fpce, Qc] = neville(h.^2,fpc,0);

format long; [fpfw(:) diag(Qfw) fpc(:) diag(Qc)]
```

0.556269565166847	0.556269565166847	0.518069447999851	0.518069447999851
0.548061072789202	0.539852580411557	0.534691718664504	0.540232475552722
0.542691210468421	0.536477604059667	0.538896367452272	0.540302279814560
0.540953537164002	0.540328764102217	0.539950615251025	0.540302305866725
0.540471706546771	0.540303517901906	0.540214370333548	0.540302305868139
0.540345469568656	0.540302304028555	0.540280321179402	0.540302305868140
0.540313197801744	0.540302305857246	0.540296809645632	0.540302305868133
0.540305041434280	0.540302305868260	0.540300931809369	0.540302305868143
0.540302991329838	0.540302305868240	0.540301962353254	0.540302305868145

and $\cos 1$=0.540302305868140...

From left to right in the above table, the numerical derivatives are calculated with the 3-point end-point formula, the extrapolated 3-point end-point formula, the 3-point mid-point formula and the extrapolated 3-point midpoint formula. We see the extremely fast convergence of the last method, and the 15 correct digits at $h = 2^{-6}$.

3 Automatic differentiation (optional)

Automatic differentiation, also known as algorithmic differentiation, is a technique for automatically augmenting a given computer program of a function[6] to compute also the derivative of the function. The idea is that any function that we can code on machine is down to the earth a rational function because only a finite sequence of the four fundamental operations $+$, $-$, $\times$, $\div$ and the function composition we can use, or more broadly any function on machine is realized as an elementary function. So, the closed-form derivative can be used and the chain rule can be applied to get the derivative of any funcion on machine. This is often superior to other approximations and plays an important role in finance and machine learning.

Due to the time limit, this topic is left for self-exploration.

[6] function here means in the mathematical sense