1. FFT AND IFFT IN MATLAB:
Use the Fast Fourier Transform to compute compute the k-th derivative of an appropriate function.

In MATLAB the interface for this function could look like
`y=fft_diff_k(N,I,f,k)`, with the following input variables:

N: number of grid points

I: interval for the grid

f: the function to differentiate

k: order of differentiation

Test you method and verify the results with an appropriate function, e.g.

$$f(x) = \cos\left(\frac{x}{16}\right)\left(1 + \sin\left(\frac{x}{16}\right)\right)$$

on $[0, 32\pi]$.

Possible solution:

```
% Exercise 1
function y=Exercise1(N,I,f,k)
%Gridpoints
x=linspace(I(1),I(2),N+1)'; x=x(1:N);
%Interval length & scaling factor
l=I(2)-I(1); sf=sqrt(l)/N;
%shifted and scaled versions of fft/ifft
myfft=@(x)fftshift(fft(x))*sf;
myifft=@(x)ifft(ifftshift(x))/sf;
u=f(x);
%fft transformed of u_hat
u_hat=myfft(u);
%wave numbers
lambda=(-N/2:N/2-1)'*2*pi/l;
%compute k-derivative in frequency space
y_hat=(1i*lambda).^k.*u_hat;
%transform back
y=(myifft(y_hat));
```

Listing 1: Exercise 1

```
% Exercise 1
N=128;
c=32;
x=linspace(0,c*pi,N+1);x=x(1:N)';
f=@(x)cos(x/16).*(1+sin(x/16));
fs=@(x)-1/16*sin(x/16).*(1+sin(x/16))+1/16*cos(x/16).^2;
u=f(x);
subplot(2,2,1), plot(x,u); title('function');
subplot(2,2,2), plot(x,fs(x)); title('first derivative');
%fft transformed of u_hat
v=Exercise1(N,[0,32*pi],f,1);
subplot(2,2,4), plot(x,v); title('first derivative -fft');
subplot(2,2,3), semilogy(x,abs(v-fs(x))); title('absolute error');
norm(v-fs(x))
```

Listing 2: Example use of Exercise 1

2. Computing φ-functions via the matrix exponential

   Implement a function that computes

$$y = \sum_{\ell=0}^{p} \tau^{\ell} \varphi_{\ell}(\tau A) v_{\ell}.$$

   See the excerpt of [Al-Mohy and Higham 2011, Ch. 2] in the appendix. In particular formula (2.11) should be implemented. Be aware of the order of the vectors v_ℓ.

   Possible solution:

```
function y=phi(tau,A,v)

p=size(v,2)-1;
J=spdiags(ones(p,1),1,p,p);
W=v(:,end:-1:2);
nw=norm(W,inf);
eta = 2^-min(ceil(log2(max(nw,realmin))));
B = [A,eta*W;zeros(p,size(A,2)),J];
ep=flipud(eye(p,1));
v0=[v(:,1); ep/eta];
y=full([speye(size(A)),sparse(length(A),p)]*expm(tau*B)*v0);
```

Listing 3: Exercise 2

For the following PDEs in one space dimension we use periodic boundary conditions in $[0, 2\pi]$ and as initial value one can use $u_0(x) = \mathrm{e}^{-100(x-3)^2}$. Discretise the space with N (even) grid points in the same fashion as in the first example. As a first iteration $N = 128$ is a fine enough grid. As final time use $T = 1$.

3. Linear PDEs

   (a) Solve the advection (transport) equation

$$\partial_t u = c_1 \partial_x u$$

   exactly by the formula

$$u(t) = u_0(x + tc_1)$$

   and with the help of the previous exercises by a FFT approach. For the experiments one can use e.g. $c_1 = \pi$ to move the initial condition for a half turn.

   (b) Use the same FFT approach to solve the heat equation

$$\partial_t u = c_2 \partial_x^2 u.$$

   For the experiment one can use $c_2 = 0.4$ as a first test.

   (c) Solve the advection-diffusion equation

$$\partial_t u = c_1 \partial_x u + c_2 \partial_x^2 u$$

   with FFT. Use c_1, c_2 as before.

   Possible solution:

```
%Configuration
N=256; I=[0,2*pi];
x=linspace(I(1),I(2),N+1)'; x=x(1:N);
%Interval length & scaling factor
l=I(2)-I(1); sf=sqrt(l)/N;
%shifted and scaled versions of fft/ifft
myfft=@(x)fftshift(fft(x))*sf;
myifft=@(x)real(ifft(ifftshift(x)))/sf;
%Parameters
c=[0.9,0.1]; T=10;
u0=@(x)exp(-100*(x-3).^2);
%timestep for animation
h=(I(2)-I(1))/N/4;

%%% Animated compuation %%%%%%%%
%%% advection exact
%disp('advection exact');
%for t=0:h:T
%    v=u0(mod(x+c(1)*t,2*pi));
%    plot(x,v), ylim([0,1]); pause(0.01);
%end
%%% advection fft
disp('advection fft');
%pause
v=u0(x);
lambda=(-N/2:N/2-1)'*2*pi/l;
L=exp(c(1)*h*1i*lambda);%+c(2)*h*(1i*k).^2);
for t=0:h:T
    v=myifft(L.*myfft(v));
    plot(x,v); ylim([-0.1,1]); pause(0.01);
end
%%% diffusion fft
disp('diffusion fft');
%pause
v=u0(x);
L=exp(c(2)*h*(1i*lambda).^2);
for t=0:h:T
    v=myifft(L.*myfft(v));
    plot(x,v); ylim([0,1]); pause(0.01);
end
%%% advection-diffusion fft
disp('advection-diffusion fft');
pause
v=u0(x);
L=exp(c(1)*h*1i*lambda+c(2)*h*(1i*lambda).^2);
for t=0:h:T
    v=myifft(L.*myfft(v));
    plot(x,v); ylim([0,1]); pause(0.01);
end
```

Listing 4: Exercise 3

4. Lie and Strang splitting

Implement a method for the Lie splitting

$$u_1 = \mathrm{e}^{\tau B}\mathrm{e}^{\tau A}u_0$$

and Strang splitting

$$u_1 = \mathrm{e}^{\frac{1}{2}\tau A}\mathrm{e}^{\tau B}\mathrm{e}^{\frac{1}{2}\tau A}u_0.$$

Assume that no dense output is required and optimise Strang splitting accordingly.

Use these implementations and test them for the *phenomenon* splitting of the advection-diffusion equation

$$\partial_t u = \underbrace{c_1 \partial_x}_{=:B} u + \underbrace{c_2 \partial_x^2}_{=:A} u.$$

As a second example test your implementation for the advection-diffusion-reaction equation

$$\partial_t u = \underbrace{\left(c_1 \partial_x + c_2 \partial_x^2\right)}_{=:A} u + \underbrace{g(u)}_{=:B}$$

where for the nonlinearity can be with the exact flow or by a Runge–Kutta method like (`ode45`). Use $g(u) = b$ the constant function and $g(u) = (1-u)u$ for your experiments.

Possible solution:

```
function y=lie(h,A,B,u0,T)
step=@(v)B(h,A(h,v));
y=u0;
for t=h:h:T
    y=step(y);
end
```

Listing 5: Lie

```
function y=strang_naive(h,A,B,u0,T)
step=@(v)A(h/2,B(h,A(h/2,v)));
y=u0;
for t=h:h:T
    y=step(y);
end
```

Listing 6: Strang naive

```
function y=strang(h,A,B,u0,T)
step=@(v)B(h,A(h,v));
y=B(h,A(h/2,u0));
for t=h:h:(T-h)
    y=step(y);
end
y=A(h/2,y);
```

Listing 7: Strang

5. Exponential Euler

Implement an exponential Euler method

$$u_1 = \mathrm{e}^{\tau A} u_0 + \tau \varphi_1(\tau A) g(u_0)$$

with the help of Exercise 2 where we implemented a function to compute linear combinations of φ-functions. As a test example use

$$\partial_t u = \underbrace{\left(c_1 \partial_x + c_1 \partial_x^2\right)}_{=:A} u + g(u)$$

for g as in the previous example i.e. $g(u) = b$ and $g(u) = (1-u)u$.

Possible solution:

```
function y=expEuler(h,A,g,u0,T)
y=u0;
for t=h:h:T
    y=phi(h,A,[y,g(y)]);
end
```

Listing 8: Exponential Euler

6. Exponential Runge–Kutta

Implement the exponential Runge–Kutta method of order two given by the Butcher tableau

$$\begin{array}{c|cc} 0 & & \\ 1 & \varphi_1 & \\ \hline & \varphi_1 - \varphi_2 & \varphi_2 \end{array}$$

or a single step as

$$\begin{aligned} u_1 &= \mathrm{e}^{\tau A} u_0 + \tau\varphi_1(\tau A) g(u_0) + \tau^2 \varphi_2(\tau A) \left(\frac{g(U_1) - g(u_0)}{\tau} \right) \\ U_1 &= \mathrm{e}^{\tau A} u_0 + \tau\varphi_1(\tau A) g(u_0) \end{aligned}$$

As a test equation use the same equations as in the previous exercise.

Possible solution:

```
function y=exprk2(h,A,g,u0,T)
y=u0;
for t=h:h:T
   y=step(h,A,g,y);
end

    function y=step(h,A,g,u0)
        gu=g(u0);
        U1=phi(h,A,[u0,gu]);
        y=phi(h,A,[u0,gu,(g(U1)-gu)/h]);
    end
end
```

Listing 9: Exponential Runge–Kutta of order 2

Possible solution:

```
%Configuration
N=128; I=[0,2*pi];
x=linspace(I(1),I(2),N+1)'; x=x(1:N);
%Interval length & scaling factor
l=I(2)-I(1); sf=sqrt(l)/N;
%shifted and scaled versions of fft/ifft
myfft=@(x)fftshift(fft(x))*sf;
myifft=@(x)real(ifft(ifftshift(x)))/sf;
lambda=(-N/2:N/2-1)'*2*pi/l;
%Parameters
c=[0.9,0.1,0.1];  T=1;
u0=@(x)exp(-100*(x-3).^2);
%timestep for animation
h=T/round(T/((I(2)-I(1))/N/4));

v=u0(x);
g=@(v)c(3)*(v.*(1-v));
g_ei=@(v)myfft(c(3)*(myifft(v).*(1-myifft(v))));
A_op=@(h,v)myifft(exp(c(1)*h*1i*lambda+c(2)*h*(1i*lambda).^2).*myfft(v));
A=spdiags((c(2)*(1i*lambda).^2+c(1)*(1i*lambda)),0,N,N);
```

```
options = odeset('RelTol',1e-13,'AbsTol',1e-15);
B_op=@(h,v)deval(ode45(@(t,x)g(x),[0 h],v,options),h);
odefun=@(t,y)myifft((c(2)*(1i*lambda).^2+c(1)*(1i*lambda)).*myfft(y))+g(y);
yode=deval(ode45(odefun,[0,T/2,T],v,options),T);
H=2.^(-1*(0:1:10));
e=zeros(length(H),4);
for i=1:length(H);
    h=H(i);
    e(i,1)=norm(yode-lie(h,A_op,B_op,v,T),inf);
    e(i,2)=norm(yode-strang(h,A_op,B_op,v,T),inf);
    e(i,3)=norm(yode-myifft(expEuler(h,A,g_ei,myfft(v),T)),inf);
    e(i,4)=norm(yode-myifft(exprk2(h,A,g_ei,myfft(v),T)),inf);
end
loglog(H,e)
legend('lie','strang','expEuler','exprk2')
```

Listing 10: Order plot for Lie, Strang, expEuler, exprk2

7. Solve the Kuramoto-Sivashinsky equation - Exercise

   On $[0, 32\pi]$ solve the Kuramoto-Sivashinsky equation

$$\partial_t u = -\partial_x^4 u - \partial_x^2 u - u\partial_x u$$

   for the initial value

$$u_0 = \cos\left(\frac{x}{16}\right)\left(1 + \sin\left(\frac{x}{16}\right)\right).$$

   with a splitting and exponential integrator approach on the interval $[0, 32\pi]$ for final time $T = 150$ and an appropriate step size τ.

   For the splitting select appropriate splitting operators with the Strang splitting and use the exponential Runge–Kutta method as exponential integrator.

   Hint: The nonlinearity can be solved exactly by the method of characteristicss.

References

Al-Mohy, A.H., Higham, N.J., 2011. Computing the action of the matrix exponential, with an application to exponential integrators. SIAM J. Sci. Comput. 33 (2), 488–511.

A Appendix

2. Exponential integrators: avoiding the φ functions. Exponential integrators are a class of time integration methods for solving initial value problems written in the form

$$u'(t) = Au(t) + g(t, u(t)), \quad u(t_0) = u_0, \quad t \ge t_0, \tag{2.1}$$

where $u(t) \in \mathbb{C}^n$, $A \in \mathbb{C}^{n\times n}$, and g is a nonlinear function. Spatial semidiscretization of partial differential equations (PDEs) leads to systems in this form. The matrix A usually represents the Jacobian of a certain function or an approximation of it, and it is usually large and sparse. The solution of (2.1) satisfies the nonlinear integral equation

$$u(t) = e^{(t-t_0)A}u_0 + \int_{t_0}^{t} e^{(t-\tau)A} g(\tau, u(\tau))\, d\tau. \tag{2.2}$$

By expanding g in a Taylor series about t_0, the solution can be written as [17, Lem. 5.1]

$$u(t) = e^{(t-t_0)A}u_0 + \sum_{k=1}^{\infty} \varphi_k\big((t-t_0)A\big)(t-t_0)^k u_k, \tag{2.3}$$

where

$$u_k = \frac{d^{k-1}}{dt^{k-1}} g(t, u(t)) \mid_{t=t_0}, \quad \varphi_k(z) = \frac{1}{(k-1)!}\int_0^1 e^{(1-\theta)z}\theta^{k-1}\, d\theta, \quad k \ge 1.$$

By suitably truncating the series in (2.3), we obtain the approximation

$$u(t) \approx \widehat{u}(t) = e^{(t-t_0)A}u_0 + \sum_{k=1}^{p} \varphi_k\big((t-t_0)A\big)(t-t_0)^k u_k. \tag{2.4}$$

The functions $\varphi_\ell(z)$ satisfy the recurrence relation

$$\varphi_\ell(z) = z\varphi_{\ell+1}(z) + \frac{1}{\ell!}, \quad \varphi_0(z) = e^z,$$

and have the Taylor expansion

$$\varphi_\ell(z) = \sum_{k=0}^{\infty} \frac{z^k}{(k+\ell)!}. \tag{2.5}$$

A wide class of exponential integrator methods is obtained by employing suitable approximations to the vectors u_k in (2.4), and further methods can be obtained by the use of different approximations to g in (2.2). See Hochbruck and Ostermann [15] for a survey of the state of the art in exponential integrators.

We will show that the right-hand side of (2.4) can be represented in terms of the *single* exponential of an $(n+p)\times(n+p)$ matrix, with no need to explicitly evaluate φ functions. The following theorem is our key result. In fact we will only need the special case of the theorem with $\ell = 0$.

THEOREM 2.1. *Let $A \in \mathbb{C}^{n\times n}$, $W = [w_1, w_2, \dots, w_p] \in \mathbb{C}^{n\times p}$, $\tau \in \mathbb{C}$, and*

$$\widetilde{A} = \begin{bmatrix} A & W \\ 0 & J \end{bmatrix} \in \mathbb{C}^{(n+p)\times(n+p)}, \quad J = \begin{bmatrix} 0 & I_{p-1} \\ 0 & 0 \end{bmatrix} \in \mathbb{C}^{p\times p}. \tag{2.6}$$

Then for $X = \varphi_\ell(\tau\widetilde{A})$ with $\ell \ge 0$ we have

$$X(1:n, n+j) = \sum_{k=1}^{j} \tau^k \varphi_{\ell+k}(\tau A)\, w_{j-k+1}, \quad j = 1{:}p. \tag{2.7}$$

Proof. It is easy to show that, for $k \ge 0$,

$$\widetilde{A}^k = \begin{bmatrix} A^k & M_k \\ 0 & J^k \end{bmatrix}, \tag{2.8}$$

where $M_k = A^{k-1}W + M_{k-1}J$ and $M_1 = W$, $M_0 = 0$. For $1 \le j \le p$ we have $WJ(:,j) = w_{j-1}$ and $JJ(:,j) = J(:,j-1)$, where we define both right-hand sides to

Al-Mohy, A.H., Higham, N.J., 2011. Computing the action of the matrix exponential, with an application to exponential integrators. SIAM J. Sci. Comput. 33 (2), 488–511.

be zero when $j = 1$. Thus

$$\begin{aligned} M_k(:,j) &= A^{k-1}w_j + (A^{k-2}W + M_{k-2}J)J(:,j) \\ &= A^{k-1}w_j + A^{k-2}w_{j-1} + M_{k-2}J(:,j-1) \\ &= \cdots = \sum_{i=1}^{\min(k,j)} A^{k-i}w_{j-i+1}. \end{aligned}$$

We will write $M_k(:,j) = \sum_{i=1}^{j} A^{k-i}w_{j-i+1}$ on the understanding that when $k < j$ we set to zero the terms in the summation where $i > k$ (i.e., those terms with a negative power of A). From (2.5) and (2.8) we see that the (1,2) block of $X = \varphi_\ell(\tau\widetilde{A})$ is

$$X(1:n, n+1:n+p) = \sum_{k=1}^{\infty} \frac{\tau^k M_k}{(k+\ell)!}.$$

Therefore, the $(n+j)$th column of X is given by

$$\begin{aligned} X(1:n, n+j) &= \sum_{k=1}^{\infty} \frac{\tau^k M_k(:,j)}{(k+\ell)!} = \sum_{k=1}^{\infty} \frac{1}{(k+\ell)!} \left(\sum_{i=1}^{j} \tau^i (\tau A)^{k-i} w_{j-i+1} \right) \\ &= \sum_{i=1}^{j} \tau^i \left(\sum_{k=1}^{\infty} \frac{(\tau A)^{k-i}}{(k+\ell)!} \right) w_{j-i+1} \\ &= \sum_{i=1}^{j} \tau^i \left(\sum_{k=0}^{\infty} \frac{(\tau A)^{k}}{(\ell+k+i)!} \right) w_{j-i+1} = \sum_{i=1}^{j} \tau^i \varphi_{\ell+i}(\tau A) w_{j-i+1}. \qquad \square \end{aligned}$$

With $\tau = 1$, $j = p$, and $\ell = 0$, Theorem 2.1 shows that, for arbitrary vectors w_k, the sum of matrix–vector products $\sum_{k=1}^{p} \varphi_k(A)w_{j-k+1}$ can be obtained from the last column of the exponential of a matrix of dimension $n + p$. A special case of the theorem is worth noting. On taking $\ell = 0$ and $W = [c \ 0] \in \mathbb{C}^{n\times p}$, where $c \in \mathbb{C}^n$, we obtain $X(1\colon n, n+j) = \tau^j \varphi_j(\tau A)c$, which is a relation useful for Krylov methods that was derived by Sidje [22, Thm. 1]. This in turn generalizes the expression

$$\exp\left(\begin{bmatrix} A & c \\ 0 & 0 \end{bmatrix} \right) = \begin{bmatrix} e^A & \varphi_1(A)c \\ 0 & I \end{bmatrix}$$

obtained by Saad [21, Prop. 1].

We now use the theorem to obtain an expression for (2.4) involving only the matrix exponential. Let $W(:, p-k+1) = u_k$, $k = 1\colon p$, form the matrix $\widetilde{A}$ in (2.6), and set $\ell = 0$ and $\tau = t - t_0$. Then

$$X = \varphi_0\big((t-t_0)\widetilde{A}\big) = e^{(t-t_0)\widetilde{A}} = \begin{bmatrix} e^{(t-t_0)A} & X_{12} \\ 0 & e^{(t-t_0)J} \end{bmatrix}, \tag{2.9}$$

where the columns of X_{12} are given by (2.7), and, in particular, the last column of X_{12} is

$$X(1:n, n+p) = \sum_{k=1}^{p} \varphi_k\big((t-t_0)A\big)(t-t_0)^k u_k.$$

Al-Mohy, A.H., Higham, N.J., 2011. Computing the action of the matrix exponential, with an application to exponential integrators. SIAM J. Sci. Comput. 33 (2), 488–511.

Hence, by (2.4) and (2.9),

$$
\begin{aligned}
\widehat{u}(t) &= e^{(t-t_0)A}u_0 + \sum_{k=1}^{p} \varphi_k\big((t-t_0)A\big)(t-t_0)^k\, u_k \\
&= e^{(t-t_0)A}u_0 + X(1:n, n+p) \\
&= \begin{bmatrix} I_n & 0 \end{bmatrix} e^{(t-t_0)\widetilde{A}} \begin{bmatrix} u_0 \\ e_p \end{bmatrix}.
\end{aligned}
\tag{2.10}
$$

Thus we are approximating the nonlinear system (2.1) by a subspace of a slightly larger linear system

$$
y'(t) = \widetilde{A}y(t), \quad y(t_0) = \begin{bmatrix} u_0 \\ e_p \end{bmatrix}.
$$

To evaluate (2.10) we need to compute the action of the matrix exponential on a vector. We focus on this problem in the rest of the paper.

An important practical matter concerns the scaling of $\widetilde{A}$. If we replace W by ηW we see from (2.7) that the only effect on $X = e^{\widetilde{A}}$ is to replace $X(1:n, n+1:n+p)$ by $\eta X(1:n, n+1:n+p)$. This linear relationship can also be seen using properties of the Fréchet derivative [11, Thm. 4.12]. For methods employing a scaling and squaring strategy a large $\|W\|$ can cause overscaling, resulting in numerical instability. To avoid overscaling a suitable normalization of W is necessary. In the 1-norm we have

$$
\|A\|_1 \le \|\widetilde{A}\|_1 \le \max\big(\|A\|_1, \eta\|W\|_1 + 1\big),
$$

since $\|J\|_1 = 1$. We choose $\eta = 2^{-\lceil \log_2(\|W\|_1) \rceil}$, which is defined as a power of 2 to avoid the introduction of rounding errors. The variant of the expression (2.10) that we should evaluate is

$$
\widehat{u}(t) = \begin{bmatrix} I_n & 0 \end{bmatrix} \exp\left((t-t_0)\begin{bmatrix} A & \eta W \\ 0 & J \end{bmatrix}\right) \begin{bmatrix} u_0 \\ \eta^{-1}e_p \end{bmatrix}.
\tag{2.11}
$$

Experiment 8 in Section 6 illustrates the importance of normalizing W.

Al-Mohy, A.H., Higham, N.J., 2011. Computing the action of the matrix exponential, with an application to exponential integrators. SIAM J. Sci. Comput. 33 (2), 488–511.