In [1]:
```
%reload_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt

from hw1 import *
```

Mathematical and Numerical Physics

Numerical part 1

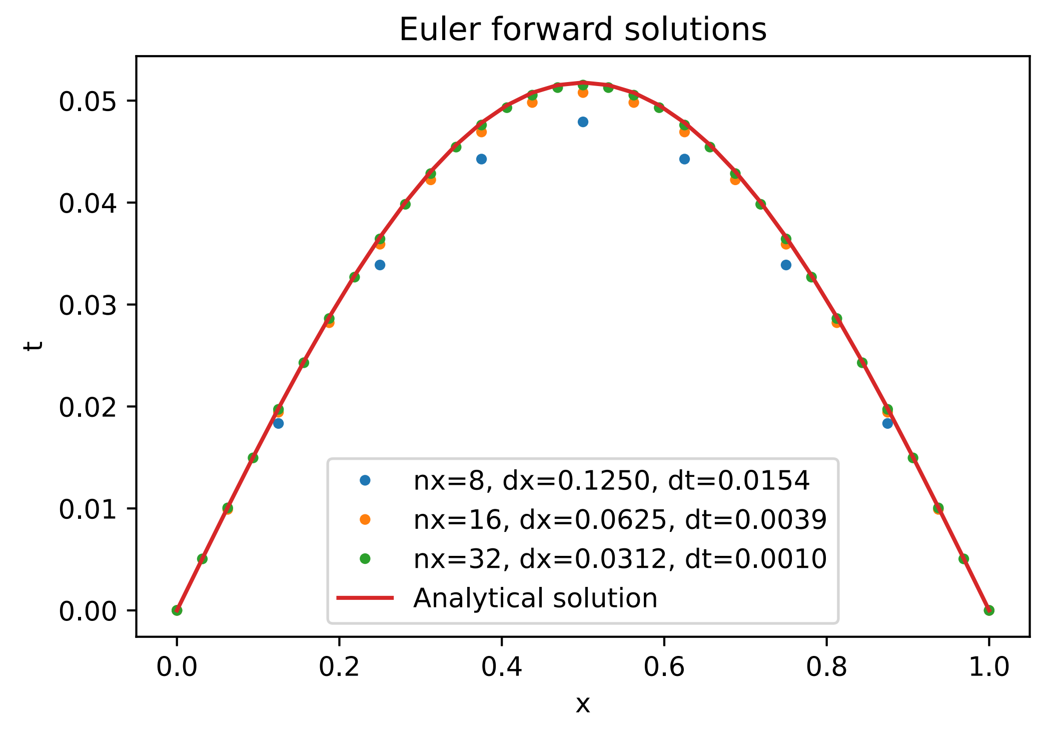
Kevin Vonk, s1706896, *Nov - Dec 2020*

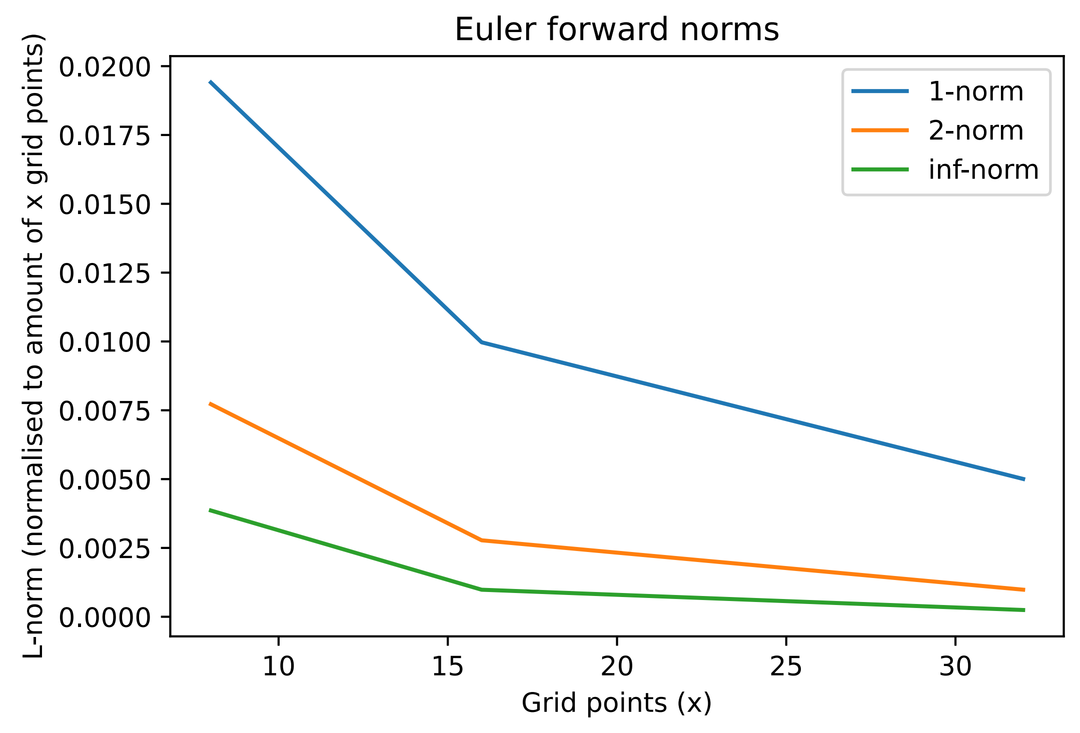
Problem 1

a.

The Euler Forward scheme has been implemented in Python using the ways discussed in the lecture notes. We will run the simulation using $n_x \in \{8, 16, 32\}$. The generating code and resulting figures are found below. Here, we will also discuss the results.

In [2]:
```
# Define some constants
ords = (1, 2, np.inf)
norms = [[] for ord in ords]

steps = (8, 16, 32)
x = (0, 1)
t = (0, 0.6)
M = 0.5
t_end = 0.6

# Compute the initial conditions and analytical solution
ic = lambda x, _: np.sin(np.pi * x)
ana = lambda x, t: np.exp(-M * np.pi**2 * t) * np.sin(np.pi * x)
err = lambda num, ana: num - ana

# Compute the solution for all given grid points in x
for nx in steps:
    nt = EulerForward1D.stable_time_steps(x, t, nx, M, nx_as_interval=True)
    ef = EulerForward1D(xbounds = x, tbounds = t, nx = nx, nt = nt, ic = ic, M = M, n

    solution = ef.fullsolve()
    error = err(solution, ana(ef.x, t[1]))
    norms = [[*norm, np.mean(np.linalg.norm(error, ord=ord))] for norm, ord in zip(no

    plt.plot(ef.x, solution, '.', label=f"nx={nx}, dx={ef.dx:.4f}, dt={ef.dt:.4f}")

# Plot the results
plt.plot(ef.x, ana(ef.x, t[1]), label="Analytical solution")
plt.legend()
plt.xlabel("x")
plt.ylabel("t")
plt.title("Euler forward solutions")

plt.figure(2)
for norm, ord in zip(norms, ords):
    plt.plot(steps, norm, label=f"{ord}-norm")

plt.legend()
plt.xlabel("Grid points (x)")
plt.ylabel("L-norm (normalised to amount of x grid points)")
plt.title("Euler forward norms");
```





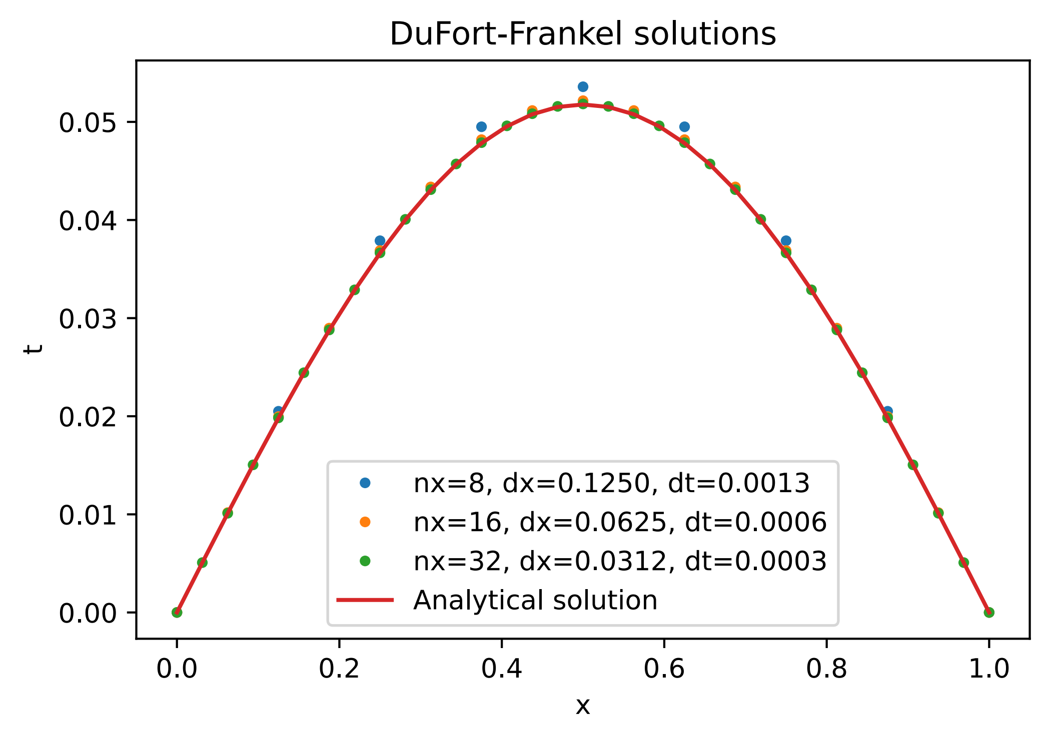
As we can observe from the numerical solutions, when we increase the amount of grid points we converge closer to the analytical solution. Note that increasing the amount of grid points also increases the amount of time grid points in order for the numerical stability to hold. The amount of grid points is computed automatically using the Von Neumann method, minimising the amount of required time grid points whilst still providing numerical stability.

The computed averaged / normalised norms show a similar picture, where the increase in grid points yield smaller errors.

b.

Now, we implement the DuFort Frankel method. We need to ensure that $\frac{\Delta t}{\Delta x} \to 0$. This can be reinterpreted as $\Delta t \ll \Delta x$. A reasonable approximation to this is taking $\Delta t = 10^{-2} \Delta x$.

In [3]:
```
# Compute the solution for the various given grid steps
for nx in steps:
    nt = DuFortFrankel1D.stable_time_steps(x, t, nx, nx_as_interval=True)
    dff = DuFortFrankel1D(xbounds = x, tbounds = t, nx = nx, nt = nt, ic = ic, M = M,
    dff.fullsolve()

    plt.plot(dff.x, dff.solution, '.', label=f"nx={nx}, dx={dff.dx:.4f}, dt={dff.dt:.4

# Plot the results
plt.plot(ef.x, ana(ef.x, t[1]), label="Analytical solution")
plt.legend()
plt.xlabel("x")
plt.ylabel("t")
plt.title("DuFort-Frankel solutions");
```



We can see from the resulting plot that the DuFort-Frankel method also nicely converges to the analytical solution. Of note here is that the DuFort-Frankel method seems to converge to the analytical solution using less positional grid points. However, the flip side of this is that our time step is about 6 times smaller, so the increased accuracy could just as well come from this decreased time step. One could argue that the factor 10^2 difference between the time steps and the positional steps needn't be this big, but there is no clear cut limit given that needs to be upheld. In any case, this solution method seems stable using these paramters and provides a good convergence to the analytical solution.

c.

In order to show that the Euler backwards method is stable, let us first consider the Euler backwards method itself,

$$-\alpha\phi_{i+1}^{n+1} + (1+2\alpha)\phi_i^{n+1} - \alpha\phi_{i-1}^{n+1} = \phi_i^n. \tag{1}$$

Applying the Von Neumann stability method to this equation means replacing ϕ_i^n with its Fourier mode,

$$\psi_i^n = \lambda^n e^{jki\Delta x}, \tag{2}$$

where j indicates the complex number, and i the positional index. Filling in eq. (2) into eq. (1) yields,

$$-\alpha\lambda^{n+1}e^{jk\Delta x}e^{jki\Delta x} + (1+2\alpha)\lambda^{n+1}e^{jki\Delta x} - \alpha\lambda^{n+1}e^{-jk\Delta x}e^{jki\Delta x} = \lambda^n e^{jki\Delta x}. \tag{3}$$

Dividing (3) by $\lambda^n e^{jki\Delta x}$,

$$\begin{aligned} -\alpha\lambda(k)e^{jk\Delta x} + (1-2\alpha)\lambda(k) - \alpha\lambda(k)e^{-jk\Delta x} &= 1 \\ \lambda(k)\left[-\alpha e^{jk\Delta x} - \alpha e^{-jk\Delta x} + 1 - 2\alpha\right] &= 1 \\ \lambda(k)\left[-2\alpha\cos(k\Delta x) - 2\alpha + 1\right] &= 1 \\ \frac{1}{1 - 2\alpha\left(\cos(k\Delta x) - 1\right)} &= \lambda(k). \end{aligned}$$

For the scheme to be stable, we have the constraint that $|\lambda(k)| \leq 1$. In this case, that means that $|1 - 2\alpha\left(\cos(k\Delta x) - 1\right)| \geq 1$. Squaring this to get rid of the absolute value and solving,

$$\begin{aligned} 1 - 4\alpha\left[\cos(k\Delta x) - 1\right] + 4\alpha^2\left[\cos(k\Delta x) - 1\right] &\geq 1 \\ 4\alpha\left[\cos(k\Delta x - 1\right]\left(\alpha\left[\cos(k\Delta x) - 1\right] - 1\right) &\geq 0 \\ 4\alpha\left[\cos(k\Delta x) - 1\right] \geq 0 \vee \alpha\left[\cos(k\Delta x) - 1\right] &\geq 0 \\ \to \alpha \geq 0 \vee \alpha &\geq \frac{1}{\cos(k\Delta x) - 1} \\ \to \alpha &\geq 0 \text{ (since } \cos(k\Delta x) \leq 1). \end{aligned}$$

So, as long as α is positive, the solution will converge. Looking at α,

$$\alpha = M\frac{\Delta t}{(\Delta x)^2},$$

we can see that Δx can be any value, and that Δt and M need to be positive. Since these two will always be positive, we can conclude that the Euler backwards method is unconditionally stable.

Problem 2

a.