2.10 Phase and Group Velocity

Given the Klein-Gordon equation

$$u_{tt} - c_0^2 u_{xx} = -q^2 u, \tag{1}$$

where $c_0 > 0$, and $q \geq 0$ constants.

Question 1

We now aim to consider solution to (1) for $x \in (-\infty, \infty)$ with initial conditions,

$$u(x,0) = f(x), u_t(x,0) = 0 \tag{2}$$

where $f(x)$ is some specified real function with $f(x) \to \infty$ as $|x| \to \infty$. For simplicity, we shall assume that f is even in x (in which case $u(x,t)$ is even in x for all t). W.L.O.G $c_0 = 1$, as if not then consider $x \to x' = Lx$. This transforms the KdV

$$u_{tt} - c_0^2 L^2 u_{x'x'} = -q^2 u.$$

So choosing $L = \frac{1}{c_0}$, we transform the KdV to (replacing x' by x now for brevity)

$$u_{tt} - u_{xx} = -q^2 u. \tag{3}$$

This transformation hasn't changed our domain for x (merely dilated our solution), and thus is valid. Thus, $c_0 = 1$ without loss of generality.

We note that when $q = 0$, we have the D'Alembert solution

$$u(x,t) = \frac{1}{2}\left[f(x-t) + f(x+t)\right]. \tag{4}$$

We shall prove something better below.

Theorem 1. *For general q, (2), (3) maybe solved by*

$$u(x,t) = \frac{1}{4\pi}\int_{-\infty}^{\infty} \tilde{f}(k) e^{ikx - i\Omega(k)t}\, dk + \text{complex conjugate(cc)}, \tag{5}$$

where $\Omega(k) = \sqrt{q^2 + k^2}$, the dispersion relation.

Proof. Fourier transform (3) in $x-$space to obtain

$$\tilde{u}_{tt} + k^2 \tilde{u} = -q^2 \tilde{u},$$

from which it follows that

$$\tilde{u}(k,t) = A(k)e^{-i\Omega(k)t} + Be^{i\Omega(k)t}$$

By our initial conditions, we have that

$$\tilde{u}(k,0) = A(k) + B(k) = \tilde{f}(k),$$

$$\tilde{u}_t(k,0) = i\Omega(k)(A(k) - B(k)) = 0,$$

thus suggesting

$$A(k) = B(k) = \frac{1}{2}\tilde{f}(k).$$

Then, by inverse fourier transform,

$$u(x,t) = \frac{1}{4\pi}\int_{-\infty}^{\infty} \tilde{f}(k)e^{ikx-i\Omega(k)t} + \frac{1}{4\pi}\int_{-\infty}^{\infty} \tilde{f}(k)e^{ikx+i\Omega(k)t}.$$

Now, since $f(x)$ is even in x, we get that its fourier transform $\tilde{f}(k)$ is a real function that is even in k. This implies that

$$\begin{aligned}\int_{-\infty}^{\infty} \tilde{f}(k)e^{i(kx+\Omega(k)t)}\,dk &= \int_{-\infty}^{\infty} \tilde{f}(-k)e^{i(kx+\Omega(k)t)}\,dk \\ &= -\int_{\infty}^{-\infty} \tilde{f}(k)e^{i(-kx-\Omega(k)t)}\,dk \\ &= \int_{-\infty}^{\infty} \tilde{f}(k)e^{-i(kx-\Omega(k)t)}\,dk\end{aligned}$$

, where the first equality follows from $\tilde{f}$ being even in k, and second equality follows from substitution $k \to -k$, and noting $\Omega(k)$ is even in k.

Now noting that $\Omega(k), \tilde{f}(k)$ are real functions, we get thus

$$u(x,t) = \frac{1}{4\pi}\int_{-\infty}^{\infty} \tilde{f}(k)e^{ikx-i\Omega(k)t}\,dk + \text{complex conjugate(cc)}.$$

$\blacksquare$

For fixed $q \geq 0$, we have that that the group velocity c_g and phase velocity c_p are given by

$$c_g = \Omega'(k) = \frac{k}{\sqrt{k^2+q^2}}$$

$$c_p = \frac{\Omega(k)}{k} = \frac{\sqrt{k^2+q^2}}{k}$$

We have plotted c_p and c_g vs k for $q > 0$ and $q = 0$ in the plots below. We note that for $q \neq 0$, $c_p, c_g \to \pm 1$ as $k \to \pm\infty$. However, $c_p \to \pm\infty$ as $k \to 0^{\pm}$, and $c_g = 0$ at $k = 0$. We further note that when $q = 0$, $c_g = c_p = \text{sgn}(k)$, and are undefined at $k = 0$.

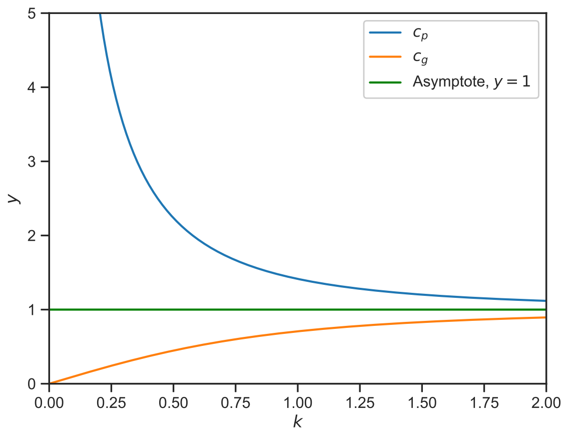


Figure 1: $q > 0$

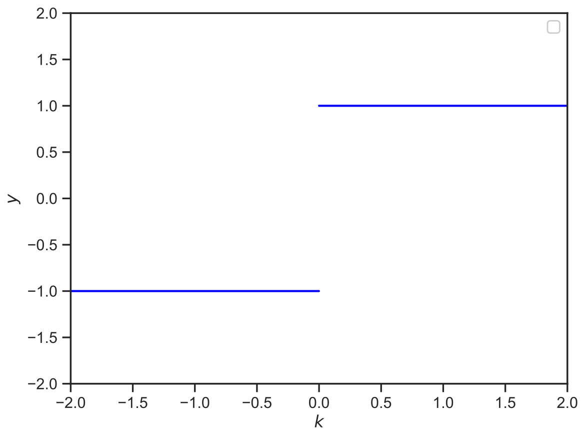


Figure 2: $q = 0$

Plots of c_p and c_g vs k

Physically, phase velocity c_p is the velocity at which crest of the wave travels (where $c_p(k)$ is the phase velocity for crests with wave number k). We note that always,

$$|c_p(k)| \geq |c_g(k)|$$

Question 2

We are told that an alternative representation of the solution in the above case is

$$u(x,t) = \frac{1}{2}\left[f(x-t) + f(x+t) - qt\int_{-\pi/2}^{\pi/2} J_1(qt\cos(\theta)f(x+t\sin(\theta))\,d\theta\right], \tag{6}$$

where J_1 is the Bessel's function of the first kind.

For large time t, we can use method of stationary phase. Consider $q > 0$, $t \to \infty$, and $V \equiv x/t$ fixed, $|V| < 1$. Let

$$I(x,t) = \int_{-\infty}^{\infty} \tilde{f}(k)e^{ikx - i\Omega(k)t}\,dk.$$

Then

$$I(Vt,t) = \int_{-\infty}^{\infty} \tilde{f}(k)e^{it(Vk-\Omega(k))}\,dk.$$

Let $\phi(k) = (Vk - \Omega(k))t$ (call it the phase). Note $\phi'(k) = (V - \Omega'(k))t$.

If $\phi'(k) \neq 0$ always, then the phase is monotonic, and thus very rapid oscillations and much cancellation (thus integral decays exponentially). Let k_0 be the solution to

$$\Omega'(k) = V.$$

However if $\Omega'(k) = V$ at $k = k_0$, then have a stationary point, and so cancellation much weaker around a small neighbourhood of k_0, thus maximum contribution to the integral comes from this small neighbourhood.

In fact we note near $k = k_0$

$$\phi(k) = \phi(k_0) - \frac{1}{2}\Omega''(k_0)t + \cdots,$$

so $\phi(k)$ changes rapidly with k again for $|k - k_0| \gg \frac{1}{\sqrt{|\Omega''(k)|t}}$, so only $|k-k_0|\,O\left(\frac{1}{\sqrt{|\Omega''(k)|t}}\right)$ matters as $t \to \infty$. This suggests that

$$I(Vt,t) \sim \tilde{f}(k_0)e^{it[Vk_0 - \Omega(k_0)]}\int_{k_0-\epsilon}^{k_0+\epsilon} e^{-it\frac{\Omega''(k)(k-k_0)^2}{2}}\,dk.$$

By standard steepest descent argument, we can replace the above and lower limits by $\pm\infty$, and we get that

$$I(Vt,t) \sim \tilde{f}(k_0)\sqrt{\frac{2\pi}{|\Omega''(k)|\,t}}e^{it[Vk_0-\Omega(k_0)] - i\mathrm{sgn}(\Omega''(k_0))\frac{\pi}{4}}.$$

We note now that

$$\Omega''(k) = \frac{q^2}{(k^2+q^2)^{3/2}},$$

and so $\Omega''(k) > 0$. Thus

$$I(Vt,t) \sim \tilde{f}(k_0)\sqrt{\frac{2\pi}{\Omega''(k)t}}e^{it[Vk_0-\Omega(k_0)] - i\frac{\pi}{4}}.$$

Using this, for $q > 0$, by method of stationary phase

$$u(x,t) \sim \frac{1}{4\pi}\tilde{f}(k_0)\sqrt{\frac{2\pi}{\Omega''(k)t}}e^{it[Vk_0-\Omega(k_0)] - i\frac{\pi}{4}} + \mathrm{cc},$$

ie

$$u(x,t) \sim \sqrt{\frac{1}{2\pi\Omega''(k)t}}\left|\tilde{f}(k_0)\right|\cos\left(t[Vk_0 - \Omega(k_0)] - \frac{\pi}{4} + \arg(\tilde{f}(k_0))\right) \tag{7}$$

If $\tilde{f}(k)$ is real, then this simplifies to

$$u(x,t) \sim \sqrt{\frac{1}{2\pi\Omega''(k)t}}\left|\tilde{f}(k_0)\right|\cos\left(t[Vk_0 - \Omega(k_0)] - \frac{\pi}{4}\right)$$

Note, now, that $\Omega'(k) = \frac{k}{\sqrt{k^2+q^2}}$, and $\tilde{f}(k)$ is even in k (so wlog $k_0 > 0$). So, for $|x| < t$:

$$k_0 = \frac{qx}{\sqrt{t^2 - x^2}}.$$

From this it follows that

$$\Omega''(k_0) = \frac{q^2}{(q^2 + k_0^2)^{3/2}} = \frac{(t^2 - x^2)^{3/2}}{qt^3}$$

and

$$Vk_0 - \Omega(k_0) = \frac{-q}{t}\sqrt{t^2 - x^2}$$

, and thus

$$u(x,t) \sim \frac{q^{\frac{1}{2}t}}{(2\pi)^{\frac{1}{2}}(t^2 - x^2)^{\frac{3}{4}}} \tilde{f}\left(\frac{qx}{\sqrt{t^2 - x^2}}\right) \cos\left(q\sqrt{t^2 - x^2} + \frac{\pi}{4}\right) \tag{8}$$

Quite clearly, V is the group velocity at wave number k_0. Throughout method of stationary phase, physically, we are analysing $u(x,t)$ as a linear superposition of waves e^{ikx} with amplitudes $\tilde{f}(k)$. When we travel at c_g, as times $t \to \infty$, the leading contribution comes from the corresponding wave number and the contributions from all other wave numbers goes to zero. So we will eventually only see those waves in the initial spectrum with wavenumber k_0. That is to say, the group velocity is the velocity of propagation of the envelope of the wave-packet whose velocity is dominated by the wave with the wavenumber k_0. This is similar to saying that the information contained in the wave is propagates at group velocity c_g.

Question 3

We use the following finite difference scheme:

$$\frac{u_i^{j+1} - 2u_i^j + u_i^{j-1}}{(\Delta t)^2} - \frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{(\Delta x)^2} = -q^2\left(\frac{u_{i+1}^j + u_{i-1}^j}{2}\right)$$

That is

$$u_i^{j+1} = 2u_i^j - u_i^{j-1} + \frac{(\Delta t)^2}{(\Delta x)^2}\left(u_{i+1}^j - 2u_i^j + u_{i-1}^j\right) - \frac{(q\Delta t)^2}{2}\left(u_{i+1}^j + u_{i-1}^j\right)$$

We have

$$u_i^{-1} = u_i^1$$

$$u_{-1}^j = u_1^j, u_N^j = 0$$

where we have taken L large enough for last condition For the first time step:

$$u_i^1 = 2u_i^0 - u_i^{-1} + \frac{(\Delta t)^2}{(\Delta x)^2}\left(u_{i+1}^0 - 2u_i^0 + u_{i-1}^0\right) - \frac{(q\Delta t)^2}{2}\left(u_{i+1}^0 + u_{i-1}^0\right)$$

i.e

$$u_i^1 = u_i^0 + \frac{(\Delta t)^2}{2(\Delta x)^2}\left(u_{i+1}^0 - 2u_i^0 + u_{i-1}^0\right) - \frac{(q\Delta t)^2}{4}\left(u_{i+1}^0 + u_{i-1}^0\right)$$

We are able to use $u_i^{-1} = u_i^1$ because one of the boundary conditions is $u_t(x,0) = 0$. From now, let $\Delta t = k, \Delta x = h$.Thus, for $x = ih, y = jk$,

$$0 = u_t(x,0) = \lim_{k \to \infty} \frac{u_i^1 - u_i^{-1}}{2k} \implies u_i^{-1} = u_i^1$$

We use the initial condition

$$f(x) = \mathbb{1}[|x| \leq 1](1 - x^2)^2.$$

This scheme was implemented in `Python 3`, and the source code can be found in the program listings in the appendix.

We note that, using fourier analysis to analyse the stability of the numerical scheme, we obtain (using $\mu = \frac{k}{h}$)

$$\hat{u}^{j+1} = -\hat{u}^{j-1} + 2\hat{u}^j\left(1 - mu^2 + \cos(\theta)\left(\mu^2 - \frac{q^2\mu^2h^2}{2}\right)\right).$$

Analysing the characteristic equation, we note that, for stability, the following must be true:

$$\left(1+\mu^2\left(-1+\cos(\theta)-\frac{q^2h^2}{2}\cos(\theta)\right)\right)^2 \leq 1$$

From this it follows that

$$-2 \leq \left(1+\mu^2\left(-1+\cos(\theta)-\frac{q^2h^2}{2}\cos(\theta)\right)\right) \leq 0,$$

and thus

$$0 \leq h \leq \frac{2}{q}$$

and

$$0 \leq \mu \leq 1$$

We run the program, firstly, for $q = 0$. We run till time $t = 50$. The plot, when run for $h = 0.01, \frac{k}{h} = 1$ is plotted below. We chose to run the program with $x-$range being $x \in [0, 60]$, as clearly it is sufficiently large such that the outer-boundary values can be safely set to 0.

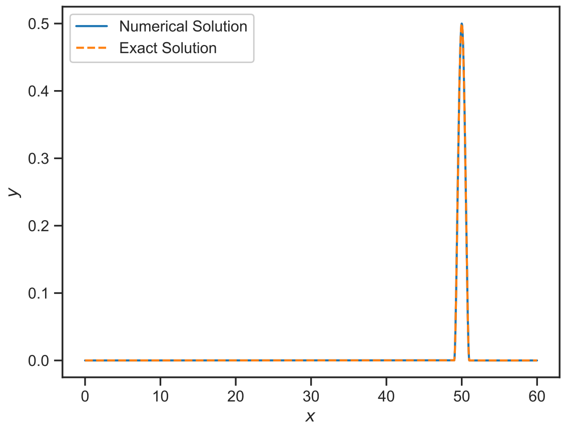


Figure 3: $q = 0, h = 0.01, \mu = 1, t = 50$

We experimented with different values of h and $\frac{k}{h}$, and recorded the following results about stability and accuracy of the numerical program below.

We note that our numerical solution is stable for $\mu \leq 1$ (as can be seen with the fact that mean and max absolute errors blow up for different values of h when μ exceeds 1 in the table below). This agrees with our fourier analysis above!

As for analysing the accuracy, we note that (concerning ourselves with $\mu \leq 1$ only) as h increases, the error increases as well (for the same value of μ), which is what we would expect! Furthermore, interestingly, it seems like the error seems to increase (and thus accuracy seems to get worse) as μ decreases below 1 (only the case of $\mu = 0.5$ is tabulated below to indicate this). In fact, the absolute errors (mean and maximum) seem to be minimised at somewhere around $\mu = 1$! As a result, taking into account number of iterations needed during computation, for $q = 0$, we use $h = 0.01, \mu = 1$!

h	$\mu = \frac{k}{h}$	Mean absolute error	Maximum absolute error
0.01	0.5	$5.291456650576513e-05$	0.004978782892600142
0.01	1	$3.174433604164178e-14$	$5.0459636469213365e-14$
0.01	1.0001	$1.091864130750299e+51$	$1.1371840661976347e+52$
0.01	1.001	$4.690174308842572e+183$	$7.12282627655415e+184$
0.1	0.5	0.003533653874143304	0.10433739010934762
0.1	1	$8.296546320322543e-16$	$4.85722573273506e-15$
0.1	1.01	$3.162691732848076e+54$	$3.2719019600302153e+55$
0.1	1.1	$8.452956343096569e+168$	$1.2780487363699053e+170$
0.6	0.5	0.03558011338721346	0.27212702868752275
0.6	0	0.006727999999999999	0.2559999999999988
0.6	1.1	$9.003217335170253e+25$	$7.347893751574626e+26$
0.6	2	$5.439300077260764e+44$	$8.80031011195271e+45$

Table 1: Tabulated values of Mean and Maximum absolute errors for different values of h, μ when $q = 0$

We, next, run the program for $q = 1$. We run till time $t = 50$. The plot, when run for $h = 0.01, \frac{k}{h} = 1$ is plotted below. We chose to run the program with $x-$range being $x \in [0, 60]$, as, once again, it is clearly sufficiently large such that the outer-boundary values can be safely set to 0.

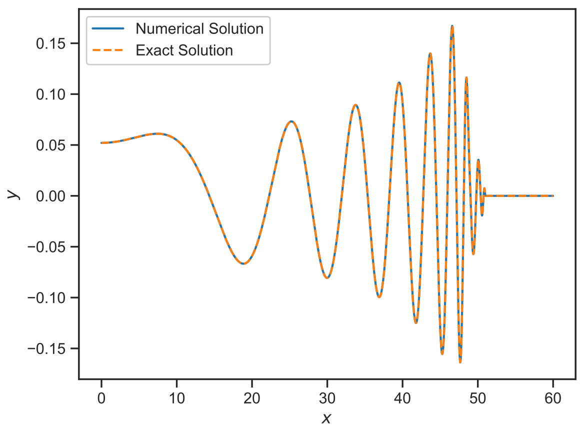


Figure 4: $q = 1, h = 0.01, \mu = 1, t = 50$

For $q = 1$, we shall use the exact solution in (6), which in this case implies that the exact solution is

$$u(x,t) = \frac{1}{2}\mathbb{1}[|x| \leq 1](1-(x-t)^2)^2 + \frac{1}{2}\mathbb{1}[|x| \leq 1](1-(x+t)^2)^2 - \\ \frac{1}{2}\int_{\max\left(-\frac{\pi}{2}, -\arcsin\left(\frac{1+x}{t}\right)\right)}^{\min\left(\frac{\pi}{2}, \arcsin\left(\frac{1-x}{t}\right)\right)} d\theta \, J_1(qt\cos(\theta))(1-(x+t\cos(\theta))^2)^2$$

We, again, experimented with different values of h and $\frac{k}{h}$, and recorded the following results about stability and accuracy of the numerical program below.

We note that our numerical solution is stable for $\mu \leq 1$ (as can be seen with the fact that mean and max absolute errors blow up for different values of h when μ exceeds 1 in the table below). Also, as opposed to having no upper bound on h in the case of $q = 0$, we seem to have an upper bound on h for stability. We, analytically, predicted this to be 2, and we note that this agrees with our numerical result below, as regardless of value of μ, the scheme seems to be clearly unstable at $h = 2.1$. This agrees with our fourier analysis above, again!

As for analysing the accuracy, we note that (concerning ourselves with $\mu \leq 1$ only, and $h \leq 2$) as h increases, the error increases as well (for the same value of μ), which is what we would expect! Furthermore, again,, it seems like the error seems to increase (and thus accuracy seems to get worse) as μ decreases below 1 (only the case of $\mu = 0.5$ is tabulated below to indicate this). In fact, the absolute errors (mean and maximum) seem to be minimised at somewhere around $\mu = 1$! As a result, taking into account number of iterations needed during computation, for $q = 1$, we use $h = 0.01, \mu = 1$ again!

h	$\mu = \frac{k}{h}$	Mean absolute error	Maximum absolute error
0.01	0.5	0.00011949427975765437	0.003912952795207056
0.01	1	$3.161753529171554e-05$	0.00042759431971278045
0.01	1.0001	$2.841380633784487e+182$	$4.309866819517182e+183$
0.1	0.5	0.00826536257688351	0.14767871688023043
0.1	1	0.003184425401806746	0.04252752688935886
0.1	1.01	$3.3956021930903245e+50$	$3.4207304469416e+51$
0.1	1.1	$5.446757194847787e+167$	$8.22990513321862e+168$
0.6	0.5	0.047610233524724016	0.2920034402612176
0.6	0	0.047036136042006696	0.3254587805709509
0.6	1.1	$2.1008464318872896e+17$	$1.536486543567048e+18$
0.6	2	$5.439300077260764e+44$	$8.80031011195271e+45$
2.1	0.5	40365.46230825442	129072.92355196078
2.1	0	35412.586565589285	217669.5273322361
2.1	1.1	298929428.4969326	1285359546.6019258
2.1	2	528195778330.5294	3984500221569.0

Table 2: Tabulated values of Mean and Maximum absolute errors for different values of h, μ when $q = 1$

We note that

$$\begin{aligned}\tilde{f}(k) &= \int_{-\infty}^{\infty} f(x)e^{-ikx}\,dx \\ &= \int_{-\infty}^{\infty} (1-x^2)^2 e^{-ikx}\,dx \\ &= -\frac{16}{k^3}\sin k - \frac{48}{k^4}\cos k + \frac{48}{k^5}\sin k\end{aligned}$$

The fourier transform $\tilde{f}(k)$ is plotted below (we note that the fourier transform $\tilde{f}(k)$ has a singularity at $k = 0$:

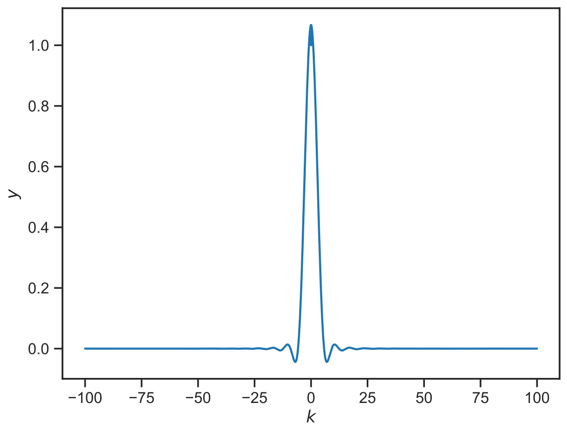


Figure 5: Plot of $\tilde{f}(k)$

We plot finite difference solutions at different times t (using $h = 0.01, \mu = 1$) below for $q = 0$ and $q = 1$, superposing the $q = 1$ case with the stationary phase approximation.

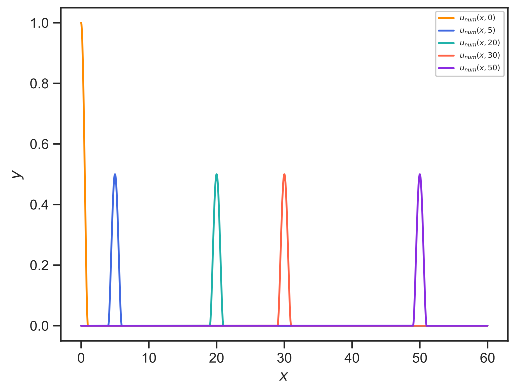


Figure 6: Plots of $u(x,t)$ (numerical) for multiple times t

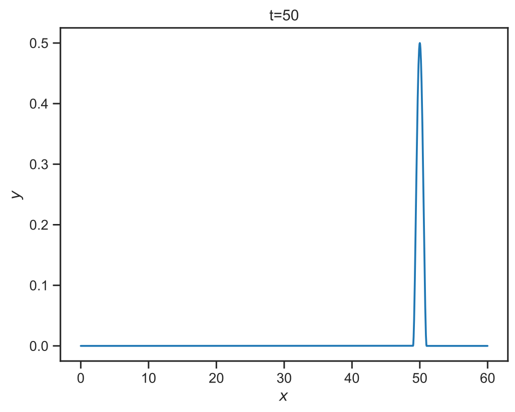
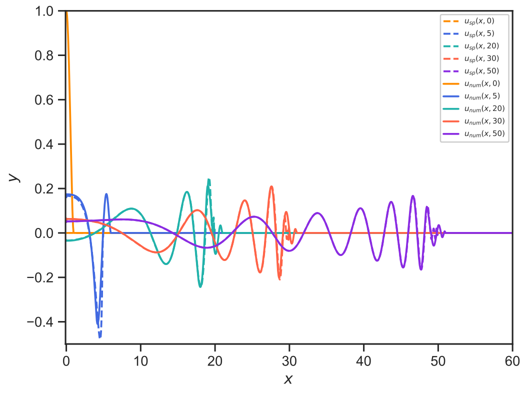


Figure 7: Plots of $u(x,t)$ for $t = 50$

Plots of solution for $q = 0$



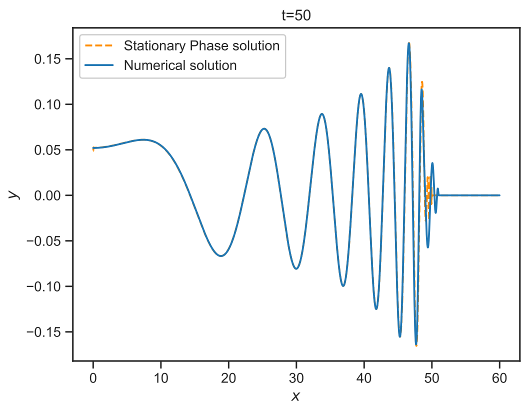


Figure 8: Plots of $u(x,t)$, both numerical and stationary phase solution for multiple times t

Figure 9: Plots of $u(x,t)$, both numerical and stationary phase solution for $t = 50$

Plots of solution for $q = 1$

We note that, as we would expect, the stationary phase approximation is more accurate for larger values of t. We also note that $u_{\text{num}}(x,0)$ is the same for $q=0$ and $q=1$ (with a bump at $x=0$). Also, both solutions have the largest peak at $t=0$, at $x=0$. Also we note that after the last bump, the solutions seem to go to 0 (in both cases, $q=0$ and $q=1$).

In the case of $q=0$, we note that the solution has a peak at $x=t$, and solutions have only one peak (and no troughs). Each peak also has the same magnitude and shape. This suggests that the wave is non-dispersive, and $\Omega(k)\propto k$, which is true! Also the single peak travels in positive x direction, at the speed $c_0=1$ (which is what we would expect!). This suggests that the group and phase velocities, both are $c_p=c_g=1$!

On the other hand, in the case of $q=1$, the number of peaks and troughs increase as t increases, and the solution has these peaks and troughs till $x=t$. We notice that their are multiple different smaller waves (like a train of waves) with different wave numbers, and thus different amplitudes. We also note that the amplitude of these individual waves seems to peak at some value of $k=k_0$. Also, the amplitude of these peaks is less than the amplitude of the peak in the case of $q=0$! As t increases, the amplitude of the individual waves decreases as well. This suggests that the wave is dispersive, and $\Omega(k)$ is not linear in and proportion to k, which is true ($\Omega(k)=\sqrt{1+k^2}$)!

In the case of $q=1$, we also note that the individual peaks seem to move in positive x direction. Furthermore, there seems to be an envelope of peaks concentrated around $k=k_0$, which moves at velocity roughly similar to that of the peak. We note that the envelope also seems to move in the positive x direction.However, the velocity of the individual waves seems to be larger than the velocity of the envelope/wave-packet. This makes sense, as the envelope travels at the group velocity c_g of the wave, and the individual waves of wavenumber k travels at phase velocity c_p. It roughly, also, seems like the envelopes are roughly sinusoidal, suggesting that it is like a Gaussian wave-packet travelling in the positive x direction.

We analyse the case of q small but not 0 in plots below. As q decreases towards 0, the total number of peaks and troughs in the solution decreases, with the number of peaks tending towards 1 and number of troughs tending towards 0. Also the amplitudes of the peaks in the solutions increases as q tends towards 0, and the amplitudes of the troughts decreases to 0 for the same limit. The plots further suggest that as $q\downarrow 0$, the solution eventually ends up with a single peak at $x=t$, and thus we note that $c_p\downarrow 1$, $c_g\uparrow 1$ in this limit. This is what we would expect from our analysis in Question 1. This suggests that the smaller the value of q, the smaller is the dispersive effect for the waves.

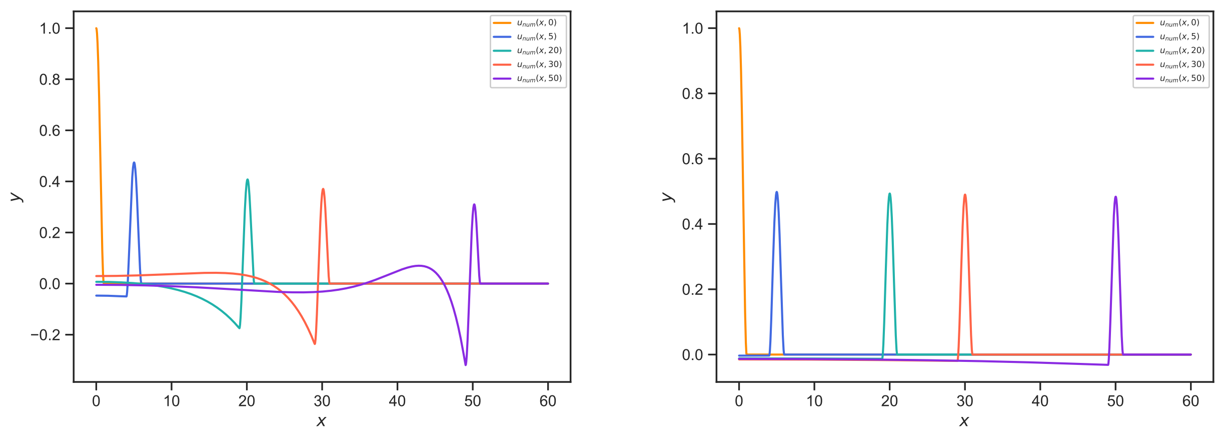


Figure 10: Plots of solution for different times t, for $q=0.2$, $q=0.01$

Question 4

We now intend on analysing the solution to (1) for $x>0, t>0$ with initial conditions

$$u(x,0)=u_t(x,0)=0$$

and boundary condition

$$u(0,t)=\sin(\omega_0 t)$$

By the same logic as in question 1, we can without loss of generality say $c_0 = 1$ in (1) (noting that dilation as in Question 1 does not change our domain, again). In this case, (1) becomes (3)

We have implemented the earlier numerical scheme to solve these initial and boundary conditions, once again, in `Python 3`. The source code can be found in the program listings in the appendix.

We first try to find the analytical solution to (3) with the above initial and boundary conditions in the case of $q = 0$. With $q = 0$, (3) becomes

$$u_{tt} - u_{xx} = 0 \tag{†}$$

Given the nature of the problem and the domain for time and space, we shall try solving it using Laplace transforms. Taking Laplace transform of (†), and noting the initial conditions, we get that (using $U(x,s)$ for the Laplace transform in time of $u(x,t)$):

$$s^2 U(x,s) - U_{xx}(x,s) = 0.$$

It thus follows that:

$$U(x,s) = A(s)e^{sx} + B(s)e^{-sx}.$$

However, since we do not want a solution that diverges as $x \to \infty$ (radiation condition) (since our domain is only $x > 0$) suggest $A(s) \equiv 0$, thus

$$U(x,s) = B(s)e^{-sx}.$$

Taking the Laplace transform of our boundary condition, we get that (letting the Laplace transform of the boundary condition be $G(s)$):

$$U(0,s) = G(s),$$

suggesting that

$$U(x,s) = G(s)e^{-sx}.$$

Recall that,

$$\mathcal{L}\{H(t-c)f(t)\} = e^{-sc}\mathcal{L}\{f(t+c)\}.$$

Thus, for our domain $t > 0, x > 0$

$$u(x,t) = H(t-x)\sin(\omega_0(t-x)).$$

We shall now compare our exact solution and numerical solutions in the case of $q = 0$ to check the accuracy and stability of the scheme. The plots for different times are given below, with the exact solution superposed onto to the numerical solution.

Comparing the exact and numerical solutions, we note that there is non-trivial behaviour only for $x < t$ in both cases. Also, we note (as tabulated below) that for $h = 0.1, \mu = 1$ (chosen for stability, reasonable accuracy and bearable computation time), we have the absolute errors (both mean and maximum) increase as t increases, but they are still very small (on the order of $O(10^{-14})$ and $O(10^{-15})$), thus suggesting the numerical solution calculated using the finite difference scheme is in very close agreement with our exact solution - suggesting that the program is accurate and works as intended! As a result, also, we shall use $h = 0.1, \mu = 1$ throughout the remain running of the program.

t	Mean absolute error	Maximum absolute error
10	$1.6257828416854636e-16$	$1.3322676295501878e-15$
50	$9.684766009987467e-16$	$7.119305145408816e-15$
150	$3.2520558511731077e-15$	$2.8400892748692286e-14$

Table 3: Mean and Maximum absolute errors in the case of $h = 0.1$, $\mu = 1$ and $q = 0$, for different values of t

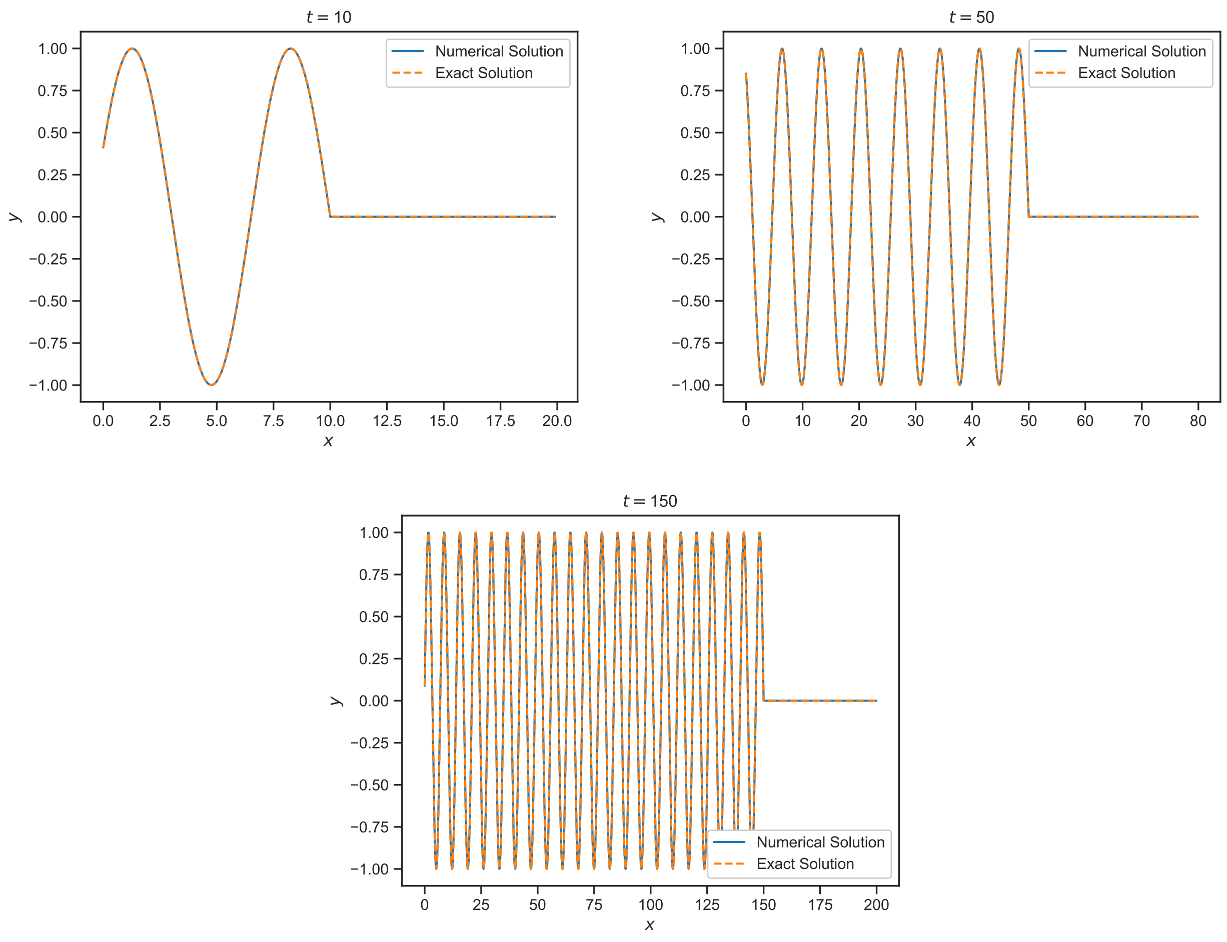


Figure 11: Plots of exact and numerical solutions for different times t, for $q = 0$

We now care about $q = 1$. We shall run till $t = 150$, and choose our domain size as $x \in [0, 200]$ as it is large enough (and more importantly, $200 > 150$) that we can set the outer-boundary values to zero safely. We have illustrated plots for $\omega_0 = 0.9, 1.1, 1.5$ below (particularly at selected times of $t = 0, 50, 100, 150$, superposed over each other).

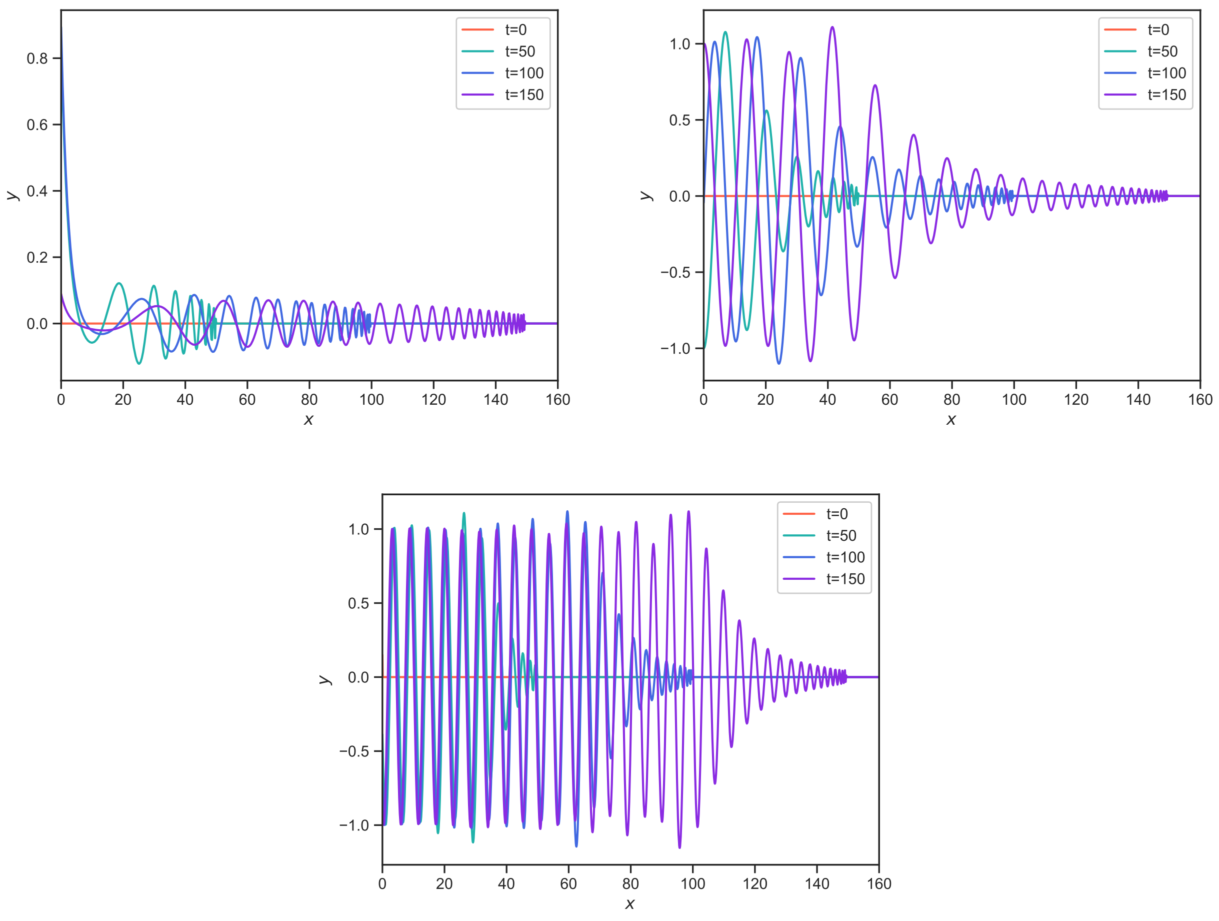


Figure 12: Plots of numerical solutions for different times t and different ω_0, for $q = 1$

From the plots above we note that the width of the envelopes increases as t increases.

In the case of $\omega_0 < 1$, particularly when $\omega_0 = 0.9$, we note that amplitude of the individual peaks is small, and thus the amplitude of the entire wave-packet is small. But this makes sense, as using our analysis in Question 2, there is much cancellation. Using this analysis, we can also conclude that the waves in the envelope are travelling out of phase.

In the case of $\omega_0 > 1$, particularly when $\omega_0 = 1.1, 1.5$, we note that amplitude of the individual packets is large (and same for the entire wave-packet), and larger ω_0, the closer the amplitudes are to 1. There is much cancellation, however, near the end, thus the exponential decay. Also, as one would expect, the higher the ω_0, the more individual waves there are in the solution.

The width of the wave-packet/envelope seems to increase as t increases in all cases. We also note that plots suggest that the group velocities c_g (the velocity of the envelope) is positive in all three cases. This is observed by noting that the group velocity will be the velocity of the peak in this envelope, which is moving in the positive $x-$ direction. Another thing we note is that as ω_0 increases, the group velocity c_g increases as well, as noted by observing the velocity of the peak in the solution. In fact, one notes that the group velocity for $\omega_0 = 1.5$ is roughly double that of $\omega_0 = 1.1$.

We also note that as ω_0 increases, the increase in the amplitude of the individual waves as t increases also increases(observe this by taking a point, say $x = 50$ in plots above, and noting the amplitude at different times t and different values of ω_0). This suggests that for waves in the envelope above (around $k = k_0$), the phase velocity eventually matches with the group velocity/velocity of the wave with wavenumber k_0, and thus becomes in phase. Thus the waves in the envelope are in phase eventually. This analysis also agrees with our analysis in Question 2.

Appendix A: Program Listings

Listing 1: Program for Question 3

```python
#---start of def---
def initial_cond_u(x,init_type=1):
    if(init_type==1):
        return ((np.heaviside(1-x,1))*(np.heaviside(1+x,1)))*((1-np.multiply
    (x,x))*(1-np.multiply(x,x)))
    else:
        return np.zeros(len(x))
#---end of def---
#---start of def---
def initial_cond_ut(x):
        return np.zeros(len(x))
#---end of def---
#---start of def---
def boundary_condition(t,w0):
    return np.sin(t*w0)
#---end of def---
#---start of def---
def initial_cond_u(x,w0=2,init_type=1):
    if(init_type==1):
        return ((np.heaviside(1-x,1))*(np.heaviside(1+x,1)))*((1-np.multiply
    (x,x))*(1-np.multiply(x,x)))
    else:
        return np.zeros(len(x))
#---end of def---
#---start of def---
def U1(u0, h, k, q):
#     up1=np.hstack([u0[1:], u0[:1]])
#     u=np.array(u0)
#     um1=np.hstack([u0[1], u0[1:]])
#     a = up1-2*u+um1
#     b=up1+um1
#     sol=u0 - ((k*k)/(2*h*h)) * a - ((q*q*k*k)/4) *  b
#     sol[-1]=0
    u=np.array(u0)
    sol=[0 for i in range(len(u0))]

    alpha=(((k*k)/(h*h))-(((q*k)*(q*k))/2))
    beta=2*(1-((k*k)/(h*h)))
    sol[0]=(alpha*u[1]+beta*u[0]+alpha*u[1])/2
    for i in range(1,len(u)-2):
            sol[i]=(alpha*u[i+1]+beta*u[i]+alpha*u[i-1])/2
    sol[-1]=0
    return sol
#---end of def---
#---start of def---
def U2(u0, u1, h, k, q):
    u=np.array(u1)
    sol=[0 for i in range(len(u1))]

    alpha=(((k*k)/(h*h))-(((q*k)*(q*k))/2))
    beta=2*(1-((k*k)/(h*h)))
    sol[0]=(alpha*u[1]+beta*u[0]+alpha*u[1])-u0[0]
    for i in range(1,len(u)-1):
            sol[i]=(alpha*u[i+1]+beta*u[i]+alpha*u[i-1]) -u0[i]
    sol[-1]=0
    return sol
#---end of def---
```

```python
#---start of def---
def solver(u0,u1,h,k,q,steps):
    for i in range(steps-1):
        U = U2(u0,u1,h,k,q)
        u0=u1
        u1=U
    return u1
#---end of def---
#---start of def---
def exact_q_0(x,t):
    return 0.5*(initial_cond_u(x-t,init_type=1)+initial_cond_u(x+t,init_type=1))
#---end of def---
#---start of def---
def exact_q_not_0_p1(x,t,q):
    ind1=np.arcsin(((1-x)/t) +0j)
    ind2=-np.arcsin(((1+x)/t) +0j)
    if(ind1.imag==0):
        b=min(np.pi/2, ind1)
    else:
        b=np.pi/2
    if(ind2.imag==0):
        a=max(-np.pi/2, ind2)
    else:
        a=-np.pi/2
    J1=integrate.quad(lambda y: (special.jv(1,q*t*np.cos(y))*(1-(x+t*np.sin(y))*(x+t*np.sin(y)))*(1-(x+t*np.sin(y))*(x+t*np.sin(y)))),a,b)
    J=0 if (x-t>1) else J1[0]
    return 0.5*(initial_cond_u(x-t,init_type=1)+initial_cond_u(x+t,init_type=1)-q*t*J)
#---end of def---
#---start of def---
def exact_q_not_0(x,t,q):
    return np.array([exact_q_not_0_p1(x_,t,q) for x_ in x])
#---end of def---
#---start of def---
def f_tilde_p1(k):
    return ((-16*((3*k*np.cos(k))+((k*k-3)*np.sin(k)))/(k*k*k*k*k))) if not(math.isnan(((-16*((3*k*np.cos(k))+((k*k-3)*np.sin(k)))/(k*k*k*k*k))))) and not(((-16*((3*k*np.cos(k))+((k*k-3)*np.sin(k)))/(k*k*k*k*k)))==0.0) else 1
#---end of def---
#---start of def---
def f_tilde(k):
    return np.array([f_tilde_p1(k_) for k_ in k])
#---end of def---
#---start of def---
def f_stationary_phase(x,t,q):
#     alpha=np.sqrt(q)*t/(np.sqrt(2*np.pi)*np.power((t*t-x*x),3/4))
#     k=q*x/(np.sqrt(t*t-x*x))
#     theta=q*np.sqrt(t*t-x*x) + (np.pi/4)
#     if(sum(x_>=t for x_ in x)>0):
#         print ("Error 1: require t>|x| for all x")
#         return False
    sol=[]
    x=list(x)
    for i in range(len(x)):
        k=q*x[i]/(np.sqrt(t**2-x[i]**2))
        if(abs(x[i])>=t*1):
            sol.append(0)
        else:
            sol.append(np.sqrt(q)*t*f_tilde([k])*np.cos(q*(np.sqrt(t**2-x[i
    ]**2))+(np.pi/4))/(np.sqrt((2*np.pi))*np.power((t**2-x[i]**2),3/4)))
    return np.array(sol)
#     return alpha*f_tilde(k)*np.cos(theta)
#---end of def---
#---start of def---
#---end of def---
#---start of def---
h=0.01
k=0.01
q=1
x=np.arange(0,60,h)
t=50
steps=int(np.ceil(t/k))
u0=initial_cond_u(x)
u1=U1(u0, h, k, q)
u_final=solver(u0,u1,h,k,q,steps)
plt.plot(x,u_final,label='Numerical Solution')
u_anal=exact_q_not_0(x,t,q)
plt.plot(x,u_anal,'--',label='Exact Solution')
plt.legend()
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.savefig('fig4.eps')
plt.show()
#==========
k=np.arange(-100,100,0.01)
ft=f_tilde(k)
plt.plot(k,ft)
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.savefig('fig5.eps')
plt.show()
#==========
h=0.01
k=0.01
x=np.arange(0,60,h)
t=50
y4=f_stationary_phase(x,t,q)
steps=int(np.ceil(t/k))
u0=initial_cond_u(x)
u1=U1(u0, h, k, q)
u_final_4=solver(u0,u1,h,k,q,steps)
t=50
plt.plot(x,y4,'--',color='darkorange',label='Stationary Phase solution')
plt.plot(x,u_final_4,label='Numerical solution')
plt.title("t="+str(t))
plt.legend()
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.savefig('fig9.eps')
plt.show()
#==========
```

Listing 2: Program for Question 4

```
#---start of def---
def initial_cond_u_2(x,init_type=1):
        return np.zeros(len(x))
#---end of def---
#---start of def---
def initial_cond_ut_2(x):
        return np.zeros(len(x))
#---end of def---
#---start of def---
def boundary_condition_2(t,w0):
    return np.sin(t*w0)
#---end of def---
#---start of def---
def initial_cond_u_2(x,w0=2,init_type=1):
        return np.zeros(len(x))
#---end of def---
#---start of def---
def U1_2(u0, h, k, q, w0):
#     up1=np.hstack([u0[1:], u0[:1]])
#     u=np.array(u0)
#     um1=np.hstack([u0[1], u0[1:]])
#     a = up1-2*u+um1
#     b=up1+um1
#     sol=u0 - ((k*k)/(2*h*h)) * a - ((q*q*k*k)/4) *  b
#     sol[-1]=0
    u=np.array(u0)
    sol=[0 for i in range(len(u0))]
    alpha=(((k*k)/(h*h))-(((q*k)*(q*k))/2))
    beta=2*(1-((k*k)/(h*h)))
    #sol[0]=(alpha*u[1]+beta*u[0]+alpha*u[1])/2
    sol[0]=boundary_condition_2(k,w0)
    for i in range(1,len(u)-2):
            sol[i]=(alpha*u[i+1]+beta*u[i]+alpha*u[i-1])/2
    sol[-1]=0
    return sol
#---end of def---
#---start of def---
def U2_2(u0, u1, h, k, q,w0,t):
    u=np.array(u1)
    sol=[0 for i in range(len(u1))]

    alpha=(((k*k)/(h*h))-(((q*k)*(q*k))/2))
    beta=2*(1-((k*k)/(h*h)))
    #sol[0]=(alpha*u[1]+beta*u[0]+alpha*u[1])-u0[0]
    #sol[0]=(alpha*u[1]+beta*u[0]+alpha*u[1])/2
    sol[0]=boundary_condition_2(t,w0)
    for i in range(1,len(u)-1):
            sol[i]=(alpha*u[i+1]+beta*u[i]+alpha*u[i-1]) -u0[i]
    sol[-1]=0
    return sol
#---end of def---
#---start of def---
def solver_2(u0,u1,h,k,q,steps,w0):
    for i in range(steps-1):
        t=k*(i+2)
        U = U2_2(u0,u1,h,k,q,w0,t)
        u0=u1
        u1=U
    return u1
```

```
#---end of def---
#---start of def---
def exact_q_0_2(x,t,w0):
    sol=[]
    for i in range(len(x)):
        if(x[i]<t):
            sol.append(boundary_condition_2((t-x[i]),w0))
        else:
            sol.append(0)
    return np.array(sol)
#---end of def---
#=========
h=0.1
k=0.1
q=0
x=np.arange(0,200,h)
t=150
w0=0.9
steps=int(np.ceil(t/k))
u0=initial_cond_u_2(x)
u1=U1_2(u0, h, k, q,w0)
u_final=solver_2(u0,u1,h,k,q,steps,w0)
plt.plot(x,u_final,label='Numerical Solution')
u_anal=exact_q_0_2(x,t,w0)
plt.plot(x,u_anal,'--',label='Exact Solution')
plt.legend()
plt.title('$t=$'+str(t))
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.savefig('fig15.eps')
plt.show()
print(max(abs(u_final-u_anal)))
print(np.mean(abs(u_final-u_anal)))
#=========
#=========
h=0.1
k=0.1
q=1
w0=0.9
x=np.arange(0,200,h)

t=0.01
steps=int(np.ceil(t/k))
u0=initial_cond_u_2(x)
u1=U1_2(u0, h, k, q,w0)
u_final=solver_2(u0,u1,h,k,q,steps,w0)
plt.plot(x,u_final,color='tomato',label='t=0')
print(x[u_final.index(max(u_final[1:]))])

t=50
steps=int(np.ceil(t/k))
u0=initial_cond_u_2(x)
u1=U1_2(u0, h, k, q,w0)
u_final=solver_2(u0,u1,h,k,q,steps,w0)
plt.plot(x,u_final,color='lightseagreen',label='t=50')
print(x[u_final.index(max(u_final[1:]))])

t=100
steps=int(np.ceil(t/k))
u0=initial_cond_u_2(x)
u1=U1_2(u0, h, k, q,w0)
```

```python
u_final=solver_2(u0,u1,h,k,q,steps,w0)
plt.plot(x,u_final,color='royalblue',label='t=100')
print(x[u_final.index(max(u_final[1:]))])

t=150
steps=int(np.ceil(t/k))
u0=initial_cond_u_2(x)
u1=U1_2(u0, h, k, q,w0)
u_final=solver_2(u0,u1,h,k,q,steps,w0)
plt.plot(x,u_final,color='blueviolet',label='t=150')
print(x[u_final.index(max(u_final[1:]))])

plt.legend()

plt.xlim(0,160)
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.savefig('fig16.eps')
plt.show()
#=========
```