B. Fourier transforms

1. In the physics convention, we may write

$$u(z,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} d\omega\ \tilde{u}(z,\omega)\exp[-i\omega t) \quad (1)$$

$$\tilde{u}(z,\omega) = \int_{-\infty}^{\infty} dt\ u(z,t)\exp(i\omega t) \quad (2)$$

Where to put the 2π is also a subject of dispute. Physicists often put in under the dt. Mathematicians of split it, with $\sqrt{2\pi}$ under the $d\omega$ and $\sqrt{2\pi}$ under the dt.

2. The Fourier transform is fundamental to linear wave propagation. Why?

In the time domain, we must solve the equation

$$i\frac{\partial u(z,t)}{\partial z} + ik_0'\frac{\partial u(z,t)}{\partial t} - \frac{1}{2}k_0''\frac{\partial^2 u(z,t)}{\partial t^2}$$

$$+ \cdots + = 0$$

somehow. In the Fourier domain, we just get

$$i\frac{\partial\tilde{u}(z,t)}{\partial z} + [k(\omega) - k_0]\hat{u}(z,t) = 0$$

which is more complete since it doesn't rely on a Taylor expansion. This equation is easily solved to yield

$$\tilde{u}(z,\omega) = \tilde{u}(0,\omega)\exp\{i[k(\omega)-k_0]z\} \quad (3)$$

$$\begin{tikzcd} u(0,t) \arrow[r, "(2)"] \arrow[d, dashed] & \tilde{u}(0,\omega) \arrow[d, "(3)"] \\ u(z,t) & \tilde{u}(z,\omega) \arrow[l, "(1)"] \end{tikzcd}$$

So, to solve the wave equation, we first determine $\tilde{u}(0,\omega)$ from $u(0,t)$ using (2). We then determine $\tilde{u}(z,\omega)$ using (3). Finally, we find $u(z,t)$ using (1).

3. When we add nonlinearity, we flip back and forth between the time and frequency domains. The key point is that derivatives are more easily evaluated in the Fourier transform domain.

4. To express the Fourier transform, we must discretize it. The usual DFT (discrete Fourier transform is written)

Note: m and n are indices, and N is the total number of points in both domains, which are therefore the same.

$$\begin{cases} \text{DFT:} & a_m = \sum_{n=0}^{N-1} \tilde{a}_n \exp(-2\pi i mn/N) \\ \text{IDFT:} & \tilde{a}_n = \sum_{m=0}^{N-1} a_m \exp(2\pi i mn/N) \end{cases}$$

Important note: In the mathematics

This is essentially a linear $N \times N$ transformation and engineering conventions, it is natural to identify DFT with the $t \to \omega$ transformation and the IDFT with the $\omega \to t$ transformation. In the physics convention, just the opposite is true.

a. Writing $\vec{A} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix}$ $\tilde{\vec{A}} = \begin{pmatrix} \tilde{a}_0 \\ \tilde{a}_1 \\ \vdots \\ \tilde{a}_{N-1} \end{pmatrix}$

we may write compactly

$$\vec{A} = DFT(\tilde{\vec{A}})$$
$$\tilde{\vec{A}} = IDFT(\vec{A})$$

Theorem: $IDFT[DFT(\tilde{\vec{A}})] = N\tilde{\vec{A}}$

Proof:

$$\tilde{a}_n = \sum_{m=0}^{N-1} a_m \exp(2\pi i m n/N)$$
$$= \sum_{m=0}^{N-1} \sum_{\ell=0}^{N-1} \tilde{a}_\ell \exp[2\pi i m(n-\ell)/N]$$
$$= \sum_{\ell=0}^{N-1} \sum_{m=0}^{N-1} \tilde{a}_\ell \exp[2\pi i m(n-\ell)/N]$$

(Exchange is easy; sum is finite)

Consider first the term $\ell = n$:

$$\sum_{m=0}^{N-1} \tilde{a}_n = N\tilde{a}_n$$

Next consider terms for which $\ell \neq n$

$$\sum_{m=0}^{N-1} \exp[2\pi i m(n-\ell)/N] = \exp[-\tfrac{2\pi i}{N}(n-\ell)/N] \cdot \sum_{m=1} \exp[2\pi i m(n-\ell)/N]$$

However, $\exp[2\pi i\, N(n-\ell)/N] = 1 = \exp[2\pi i \cdot 0 \cdot (n-\ell)/N]$

$$\Rightarrow \sum_{m=0}^{N-1} \exp[2\pi i m(n-\ell)/N] = \exp[-2\pi i(n-\ell)/N] \sum_{m=0}^{N-1} \exp[2\pi i\,(m-\ell)/N]$$

$$= 0$$

Hence, only terms for which $\ell = n$ contribute, and our result is proved ∎

b. Note that the factor N must be dealt with. In Matlab, it is put in the IDFT. Other packages deal with it differently!

5. Modern packages use the Cooley-Tukey algorithm, called the FFT to compute transforms. That reduces the operation count (number of multiplies, additions, etc.) to $N \ln N$ from N^2. The details are beyond the scope of this course, but there is one important point: In generating the transform matrix, you do not want to repeatedly calculate $e^{2\pi i mn/N}$! You want to calculate $\alpha = e^{2\pi i/N}$ and then generate the elements α^{mn} by multiplies.

This orders of magnitude faster.

In general, the same holds for:

$\cos mx = \cos x \cos[(m-1)x] - \sin x \sin[(m-1)x]$

$\sin mx = \sin x \cos[(m-1)x] + \cos x \sin[(m-1)x]$

which can be generated from $\cos x$, $\sin x$, and recursion relations.

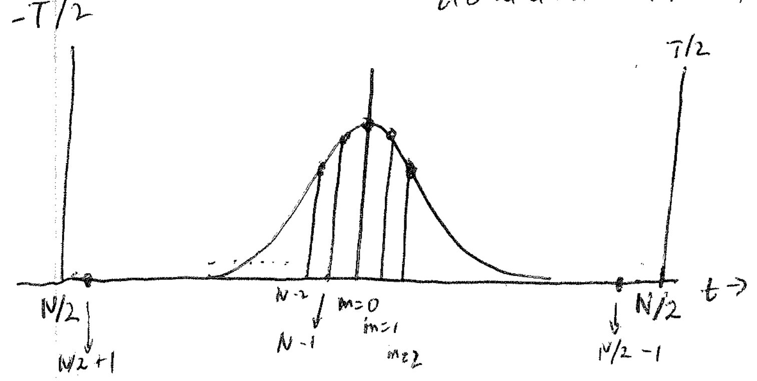
Doing it the opposite way falls into Acton's list of stupid things to do.

6. Translating from m, n to ω, t.

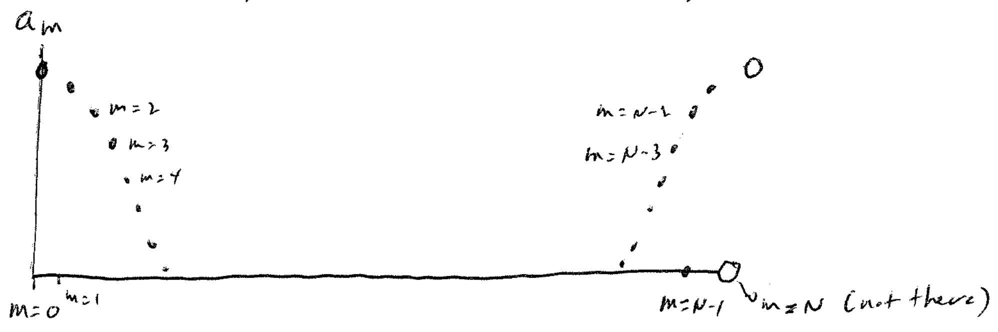
So, $\Delta t = T/N$; important point.

a. We note first that $m=0$ corresponds to $t=0$ and $n=0$ corresponds to $\omega=0$. If our domain is T in size, then we let it go from $-T/2$ to $T/2$, so t-values can be negative, while m values are not. We also note that m values are circular; $m=N$ is the same as $M=0$. So, we divide the domain as follows (assuming N is even; we usually take it to be a factor of two).

Note that the two boundaries are the same point.



Thus, when we discretize, we obtain



Stupid mistakes that you can make

① forgetting to shift the function when discretizing

② shifting the function so that it is symmetric between $m=0$ and $m=N-1$. The point $m=0$ corresponds to $t=0$, not $t=\Delta t/2$. The point $m=N-1$ corresponds to $t=-\Delta t$, not $t=-\Delta t/2$.

b. It is natural to set

$$a_m = u(z, m\Delta t), \quad 0 \le m \le N/2-1$$

$$a_m = u[z, (m-N)\Delta t], \quad N/2 \le m \le N-1$$

where $\Delta t = T/N$

Noting $\exp(2\pi i m n/N) = \exp(i\omega t)$ and using $t = m\Delta t = mT/N$, consistency implies $\omega = n\Delta\omega = n(2\pi/T)$

or: $\Delta\omega = 2\pi/T$

The full frequency domain $N\Delta\omega \equiv \Omega = 2\pi/\Delta t$

dropping z-argument Writing $\tilde{u}(\omega) = \int_{-\infty}^{\infty} dt\, u(t) \exp(i\omega t)$

$$\simeq \Delta t \sum_{m=0}^{N-1} a_m \exp(2\pi i m n/N)$$

we $\tilde{a}_n = \frac{1}{\Delta t} \tilde{u}(\omega_n)$

$$= \frac{N}{T} \begin{cases} \tilde{u}(n\Delta\omega), & 0 \le n \le N/2 - 1 \\ \tilde{u}[(n-N)\Delta\omega)], & N/2 \le n \le N \end{cases}$$

where $\Delta\omega = 2\pi/T$

The Fourier transform must be appropriately shifted as well.

c. Some packages like Matlab index from 1 to N, not 0 to $N-1$. In that case $a_1 \Leftrightarrow u(0)$, $a_2 \Leftrightarrow u(1 \cdot \Delta t)$, $a_3 \Leftrightarrow u(2 \cdot \Delta t)$, etc. You have to watch out!

C. Numerical Errors

1. When solving equations numerically, one must always discretize, which leads to errors. Monitoring these errors and keeping them below an acceptable level is a very important part of numerical computation.

   One must distinguish between <u>local</u> and <u>global</u> errors. <u>Local errors</u> are the errors made in one step. These are relatively easy to monitor. <u>Global errors</u> are the errors made in the entire computation. These are harder to monitor, but are ultimately more important.

2. We will consider the following simple example to illustrate a couple of the concepts.

   $$\frac{d^2x}{dt^2} + x = 0 \qquad [\text{harmonic oscillator}]$$

   a. First step: Reduce to two first order equations

   $$\frac{dx}{dt} = y, \quad \frac{dy}{dt} = -x$$

   General solution: $x = R\sin(t+\phi)$, $y = R\cos(t+\phi)$

6. Discretize (first order)

$$\frac{x_{n+1} - x_n}{\Delta} = y_n \qquad \frac{y_{n+1} - y_n}{\Delta} = -x_n$$

where $\Delta = t_{n+1} - t_n$, which we take to be constant. It is possible, and often useful to vary step sizes as well.

$$x_{n+1} = x_n + y_n \Delta$$

$$y_{n+1} = y_n - x_n \Delta$$

(1) Can we estimate the error?

From a Taylor expansion

$$\begin{aligned} x_{n+1} &= R\sin(t_n + \Delta + \phi) \\ &= R\sin(t_n+\phi) + \Delta R\cos(t_n+\phi) \\ &\quad - \frac{\Delta^2}{2} R\sin(t_n+\phi) - \frac{\Delta^3}{6} R\cos(t_n+\phi) + \dots \\ &= x_n + y_n\Delta - \frac{1}{2}x_n\Delta^2 - \frac{1}{6}y_n\Delta^3 + \dots \\ &= x_n + y_n\Delta + \mathcal{O}(\Delta^2) \end{aligned}$$

The $\mathcal{O}$ symbol indicates that as $\Delta \to 0$, what is left tends to a generally non-zero constant.

assuming that x_n and y_n are known. So, the method is first-order accurate, and the local error oscillates

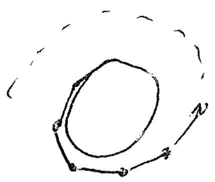
(2) What about global errors?
Discrete equations are usually harder to solve than continuous equations, but it can be done easily in this case.

We look for exponential solutions, just like in the continuous case:

$$x_n = X e^{\lambda \cdot n} = X \mu^n, \quad y_n = Y e^{\lambda n} = Y \mu^n$$

$$X\mu = X + Y\Delta, \qquad Y\mu = Y - X\mu$$

Solution: $\mu = 1 \pm i\Delta$, $Y = \pm i X$



Note that $|\mu| > 1$, so that the trajectories spiral out!

Writing $\mu = (1+\Delta^2)^{1/2} \exp\left[\pm i \tan^{-1}(\Delta)\right]$, we see that there is a phase error as well, but it is small.
$\tan^{-1}\Delta \simeq \Delta - \frac{1}{3}\Delta^3$, and it is cubic.

(3) What happens as we make our step size smaller? The local error decreases, but what about the global error?

If we wish to integrate for a total time T, the number of steps N equals T/Δ. The final radius R equals $R_0 (1+\Delta^2)^{T/2\Delta}$

$$\simeq R_0 \exp(T\Delta/2)$$

which does go to zero albeit weakly.

In general, it is considered good to use schemes that are at least second order.

C. Discretize (second order; split step)

$$x_{n+1} = x_n + y_{n+1/2}\,\Delta$$

$$y_{n+3/2} = y_{n+1/2} - x_{n+1}\,\Delta$$

We use the x and the y values at the mid-point to evaluate the derivatives. (Some scheme must be used, for example the previous scheme to advance y from y_0 to $y_{1/2}$ and $y_{N-1/2}$ to y_N).

This scheme is not general, but is very important in our applications and gives high order accuracy with the same number of function evaluations.

What are the errors? As before, we look for solutions like

$$x_n = X\mu^n, \quad y_n = Y\mu^n$$

$$\mu X = X + \mu^{1/2} Y\Delta$$
$$\mu Y = Y - \mu^{1/2} X\Delta$$
$$(\mu - 1)^2 = -\mu\Delta^2 \quad \Rightarrow \mu = \left(1 - \frac{\Delta^2}{2}\right) \pm i\left[1 - \left(1 - \frac{\Delta^2}{2}\right)^2\right]^{1/2}$$
$$Y = \pm i X$$

(1) First observation: As long as $\Delta < \sqrt{2}$, $|\mu|^2 = 1$ and there is no exponential growth! Even globally, there is no error in the amplitude. There is just phase error.

In many problems, phase error is much less important than amplitude error. That is usually the case in optical communications, Hence, the global stability of split step is a real advantage.

(2) Local error: $\mu = \left(1 - \frac{\Delta^2}{2}\right) \pm i\Delta\left(1 - \frac{\Delta^2}{4}\right)^{1/2}$

$$\simeq 1 \pm i\Delta - \frac{\Delta^2}{2} \mp i\frac{\Delta^3}{8} + \ldots$$

For the exact solution, if we write

$x_n = X\mu^n, \qquad y_n = Y\mu^n,$

we find, using $x(t) = Xe^{\pm it} = Xe^{\pm in\Delta}$

that $\mu = e^{\pm i\Delta} = 1 \pm i\Delta - \frac{\Delta^2}{2} \mp i\frac{\Delta^3}{6} + \cdots$

So, $\mu_2 - \mu_{actual} = \pm i\frac{\Delta^3}{24} + \cdots = \mathcal{O}(\Delta^3)$

where $\mu_2 = \mu$ for the second order method.
[We just proved that it is second order!]

C. Fourth-order Runge Kutta

$$x_{n+1} = x_n\left(1 - \frac{\Delta^2}{2} + \frac{\Delta^4}{24}\right) + y_n\left(\Delta - \frac{\Delta^3}{6}\right)$$

$$y_{n+1} = -x_n\left(\Delta - \frac{\Delta^3}{6}\right) + y_n\left(1 - \frac{\Delta^2}{2} + \frac{\Delta^4}{24}\right)$$

We are essentially expanding $\cos(\Delta)$ and $\sin(\Delta)$ through order Δ^4. Clearly, the error is $\mathcal{O}(\Delta^5)$

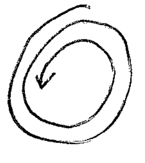
In this case, we find

$$\mu = \left(1 - \frac{\Delta^2}{2} + \frac{\Delta^4}{24}\right) \pm i\left(\Delta - \frac{\Delta^3}{6}\right)$$

$$|\mu|^2 = \left(1 - \frac{\Delta^6}{72} + \frac{\Delta^8}{576} + \cdots\right)$$

So, there is an inward spiral.

The number of revolution needed to see a change falls fast with Δ.
When $\Delta = 1$ ($\simeq 57°$) a 1% change is visible in one step.
When $\Delta = 0.1$ ($\simeq 5.7°$) 11,460 revolutions around the circle are needed;
$\left[N_{cycle} = \frac{\Delta}{2\pi} \frac{\ln 0.99}{\ln(1-\Delta^6/72)} \simeq \frac{0.0172}{2\pi\Delta^5} = 11{,}460 \right]$



D. Demo 1A : Dispersion of a Gaussian pulse

1. This problem can be done analytically

a. Basic set up: Pulses are normally specified in terms of their peak power: (A^2)

$$|u(z=0,t)|^2 = \text{power [e.g., mW]}$$

and their FWHM

We write a Gaussian pulse as

$$u(0,t) = A\exp(-t^2/2t_0^2)$$

$$|u(0,t)|^2 = A^2\exp(-t^2/t_0^2)$$

$$|u(0,\tfrac{1}{2}t_{FWHM})|^2 = A^2\exp(-t_{FWHM}^2/4t_0^2) = \frac{A^2}{2}$$

$$\Rightarrow t_0 = t_{FWHM}/2(\ln 2)^{1/2}$$

$$\simeq t_{FWHM}/1.665 \simeq 0.6006\, t_{FWHM}$$

For a pulse with $t_{FWHM} = 20$ ps, $t_0 \simeq 12$ ps.

b. First leg: $u(0,t) \rightarrow \tilde{u}(0,\omega)$

$$\tilde{u}(0,\omega) = A\int_{-\infty}^{\infty} dt\, \exp(-t^2/t_0^2)\exp(i\omega t)$$

$$= \sqrt{2\pi}\, t_0\, A\exp(-\omega^2 t_0^2/2)$$

$|\tilde{u}(0,\omega)|^2$ has units of power $\times$ time^2. To obtain power spectral density, which has units of power $\times$ time, we must divide by the repetition rate of the pulses. Suppose $T = 100$ ps and $A^2 = 1$ mW,

then $\frac{1}{T}|\tilde{u}(0,\omega)|^2$ has a peak of

$$2\pi \cdot \frac{144\ ps^2}{100\ ps} \cdot 1\ mW \cdot \boxed{2\pi \text{ radians/cycle}}$$

important conversion; easy to miss

$$\simeq 57\ ps \cdot 1\ mW = 57\ \mu W/Ghz$$

Bandwidth:

$\Delta\omega_{FWHM}$
$= \frac{1.665}{t_0}$
$= \frac{1.665}{0.012}$ Ghz

$\Rightarrow \Delta\nu_{FWHM}$
$= \frac{1.665}{2\pi \times 0.012}$ Ghz

$\simeq 22$ Ghz

c. Second leg: $\tilde{u}(0,\omega) \rightarrow \tilde{u}(z,\omega)$

$$\tilde{u}(z,\omega) = \tilde{u}(0,\omega)\exp\left(i\frac{k_0''}{2}\omega^2 z\right)$$

Normally, we measure dispersion (D) in units of ps/nm-km

k_0'' is typically in units of ps^2/km

$D = -1$ ps/nm-km
$\rightarrow k_0'' \simeq 1.2$ ps^2/km

Noting $k_0'' = \frac{d}{d\omega}k_0' = \frac{d\lambda}{d\omega}\frac{dk_0'}{d\lambda}$

$= -\frac{2\pi c}{\omega_0^2} D$ Note change of sign!

$= -\frac{\lambda_0}{\omega_0} D$

$\simeq -1.19\ [D \text{ in ps/nm-km}] \cdot$ ps^2/km

$\lambda_0 = 1.5 \times 10^3$ nm

$\omega_0 = 2\pi c/\lambda_0 = \frac{2\pi \times 3\times 10^5\ nm/ps}{1.5\times 10^3\ nm}$
$= 1.257 \times 10^3\ ps^{-1}$

d. Third leg: $\tilde{u}(z,\omega) \rightarrow u(z,t)$

$$u(z,t) = \int_{-\infty}^{\infty} d\omega \frac{t_o A}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}(t_o^2 - i k_o'' z)\omega^2\right] \cdot \exp(-i\omega t)$$

$$= \frac{A\, t_o}{(t_o^2 - i k_o'' z)^{1/2}} \exp\left[-\frac{t^2}{2(t_o^2 - i k_o'' z)}\right]$$

The $\pi/4$ phase shift when z is large is apparent from the denominator.

Analyzing the phase variation is a bit complex, but the ~~pu~~ pulse duration is easy to find:

$$|u(z,t)|^2 = \frac{A^2 t_o^2}{\left[t_o^4 + (k_o'' z)^2\right]^{1/2}} \cdot \exp\left\{-\frac{t_o^2 t^2}{\left[t_o^4 + (k_o'' z)^2\right]}\right\}$$

$$t_{FWHM}(z) = 1.665\left[t_o^2 + (k_o'' z)^2/t_o^2\right]^{1/2}$$

z is small; growth is quadratic
z is large; growth is linear

Pulses that are initially narrowest spread fastest because their bandwidths are larger and they suffer more dispersion.

1.4.4

With $t_{FWHM}(0) = 20$ ps and $D = -1$ ps/nm-km,
we have $t_0 = 12$ ps and $k_0'' = 1.2$ ps^2/km

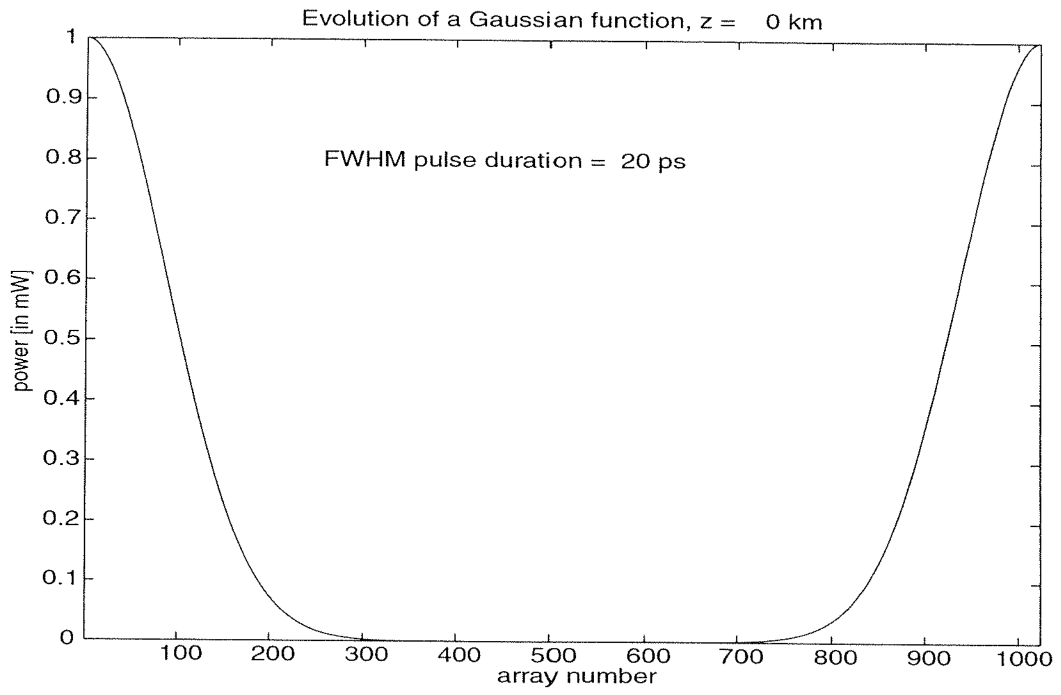
At $z = 100$ km; $t_{FWHM}(z) = 26$ ps
$z = 200$ km; $t_{FWHM}(z) = 39$ ps
$z = 500$ km; $t_{FWHM}(z) = 86$ ps
We are in the regime of linear growth
by $z = 500$ km

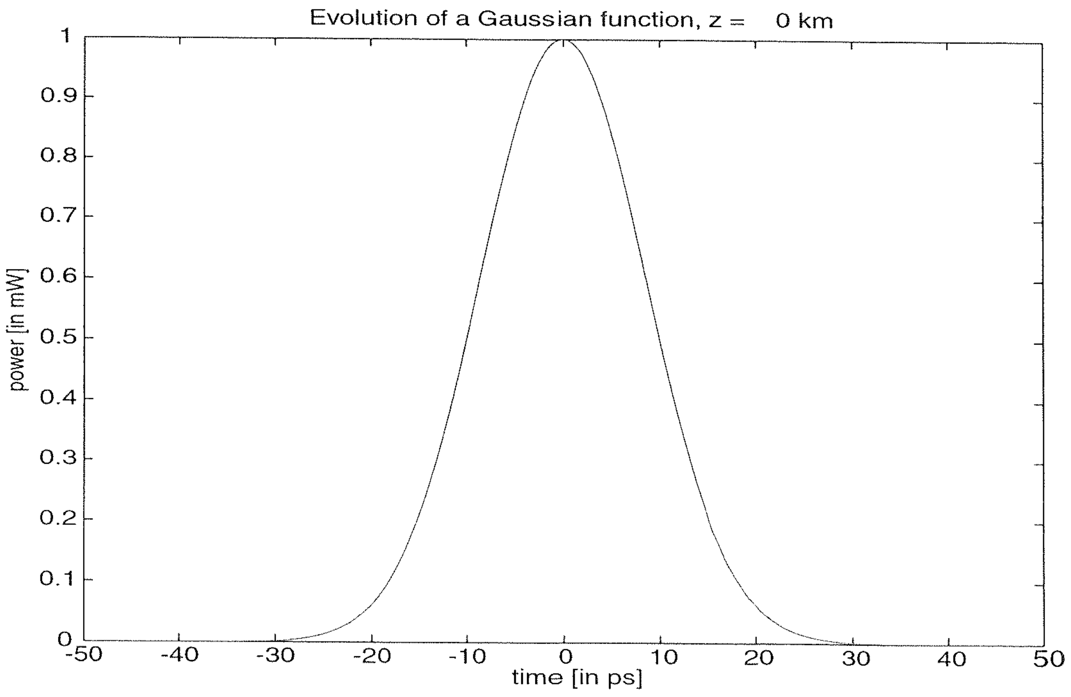
What follows is a MATLAB code and its output the shows the shift in the time domain and illustrates the perils of aliasing

MATLAB code for evolution of a Gaussian pulse in optical fibers
(January 29, 2003 — C. R. Menyuk)

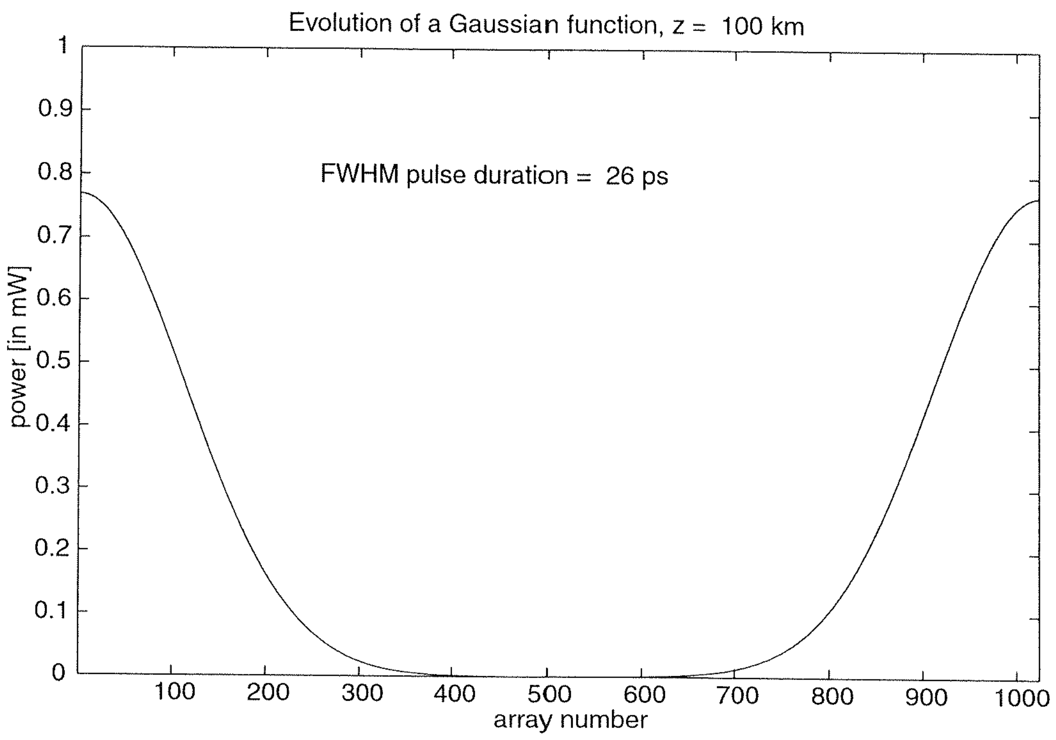
```
% Gauss_evolve
%
% Calculate and plot the evolution in an optical fiber of a Gaussian pulse

% Input parameters
%
N = 1024            % total number of points kept in time window
T_domain = 100      % total time domain kept [in ps]
t_FWHM_0 = 20       % initial pulse FWHM [in ps]
P_0 = 1             % initial peak power [in mW]
D = -1              % dispersion coefficient [in  ps/nm-km]
z_plot = [0 100 200 500]  % z-values to plot [in km]

% Derived parameters
%
Delta_t = T_domain/N;         % node spacing in time
Delta_om = 2*pi/T_domain;     % node spacing in radial frequency
Amp = sqrt(P_0);              % initial amplitude of the Gaussian
t_0 = t_FWHM_0/(2*sqrt(log(2)));    % initial pulse standard deviation
k_0_dp = -1.2*D;  % translation to ps^2/km
                  % WARNING:  Only valid at 1.5 microns

% Input vectors
%
t_vec = Delta_t*(-N/2:1:(N/2)-1);    %create array of time points
om_vec = Delta_om*(-N/2:1:(N/2)-1);  %create array of radial frequency points
u_vec = Amp*exp(-t_vec.^2/(2*t_0^2));  %create array of initial amplitudes

% Shift u-vec for the DFT
%
a_vec(1:N/2) = u_vec((N/2)+1:N);
a_vec((N/2)+1:N) = u_vec(1:N/2);

% Shift and square om_vec for use in calculating z-evolution
%
om_vec_square_shift(1:N/2) = om_vec((N/2)+1:N).^2;
om_vec_square_shift((N/2)+1:N) = om_vec(1:N/2).^2;

% Carry out IFFT (appropriate for physics convention)
%
a_tilde_vec = ifft(a_vec,N);

% Enter loop to determine and plot z-evolution
%
n_plot = length(z_plot);   %determine number of z-values
for i_plot = 1:n_plot;

    % Determine the z-value and the exponential factor for evolution
    %
    z_val = z_plot(i_plot);
    factor = exp((sqrt(-1)/2)*k_0_dp*om_vec_square_shift*z_val);
       % NOTE:  I use sqrt(-1) for i to avoid problems with re-definition

    % Carry out the z-shift and transform to time domain
    %
    a_tilde_vec_z = a_tilde_vec.*factor;
    a_vec_z = fft(a_tilde_vec_z,N);

    % Calculate powers as a function of array number and time
    %
    P_a_z = a_vec_z.*conj(a_vec_z);   % power as a function of array number
    P_u_z(1:N/2) = P_a_z((N/2)+1:N);  % shift to obtain power as a ...
    P_u_z((N/2)+1:N) = P_a_z(1:N/2);  % function of time

    % Calculate analytical formula for the power for comparison
    %
    t_z = sqrt(t_0^2 + (k_0_dp*z_val)^2/t_0^2);  % new standard deviation
    t_FWHM_z = 2*sqrt(log(2))*t_z;                % new FWHM duration
    P_comp_z = P_0*(t_0/t_z)*exp(-t_vec.^2/t_z^2);

    % Plot results
    %
    plot(P_a_z)                       % power as a function of array number
    axis([1 N 0 P_0])
    xlabel('array number')
    ylabel('power [in mW]')
    titlabel = sprintf('Evolution of a Gaussian function, z = %4.0f km'...
        ,z_val);
    title(titlabel)
    timetext = sprintf('FWHM pulse duration = %3.0f ps',t_FWHM_z);
    text(0.25*N, 0.8*P_0,timetext)
    pause
    %
    plot(t_vec,[P_u_z;P_comp_z])      % power as a function of time
    axis([-T_domain/2 T_domain/2 0 P_0])
    xlabel('time [in ps]')
    ylabel('power [in mW]')
    titlabel = sprintf('Evolution of a Gaussian function, z = %4.0f km'...
        ,z_val);
    title(titlabel)
    % --- suppress plotting of t_FWHM when it will overlap plot
    if t_0/t_z < 0.8
        timetext = sprintf('FWHM pulse duration = %3.0f ps',t_FWHM_z);
        text(-0.25*T_domain, 0.8*P_0,timetext)
    end

    pause
end
```

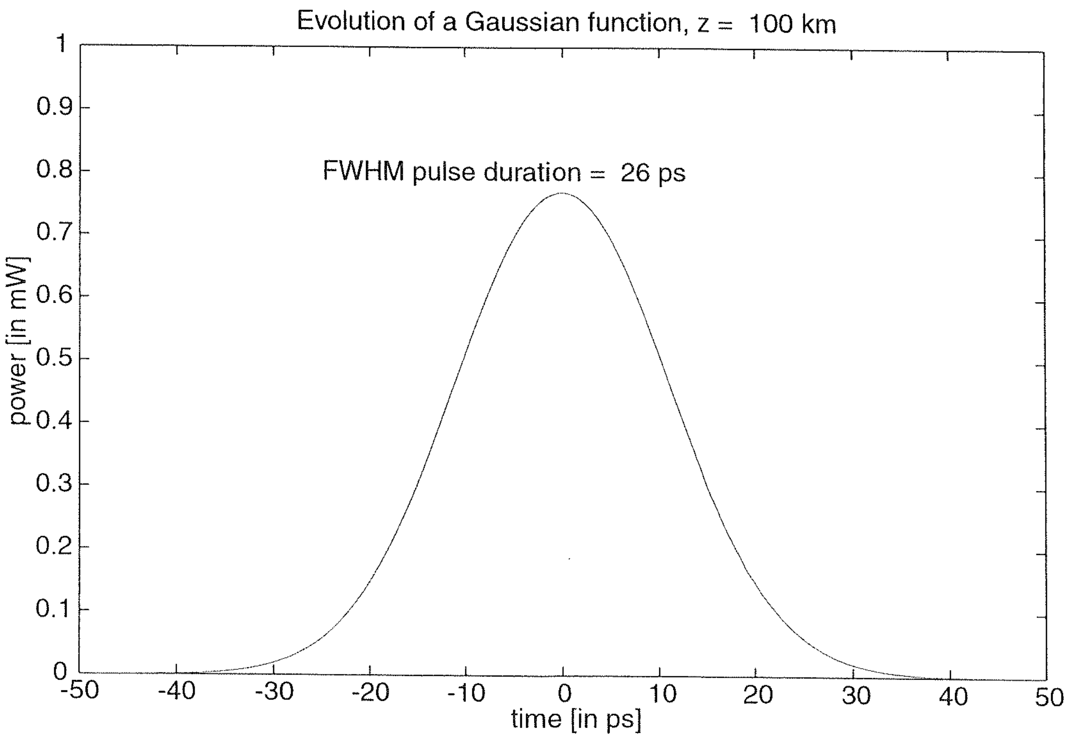
MATLAB output $z = 0$ km



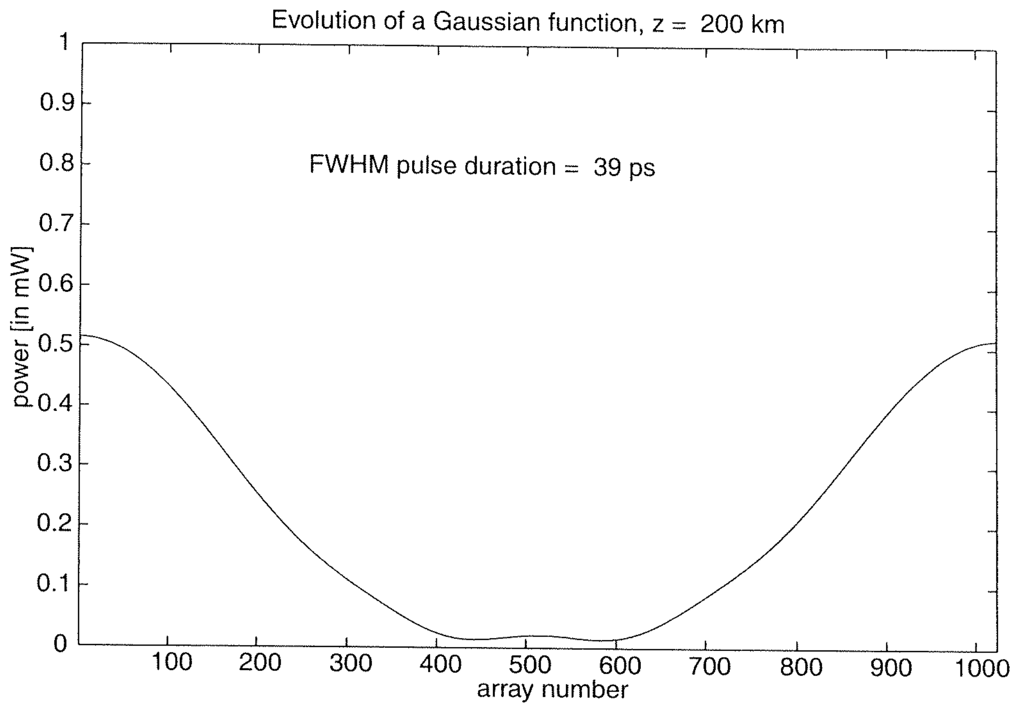


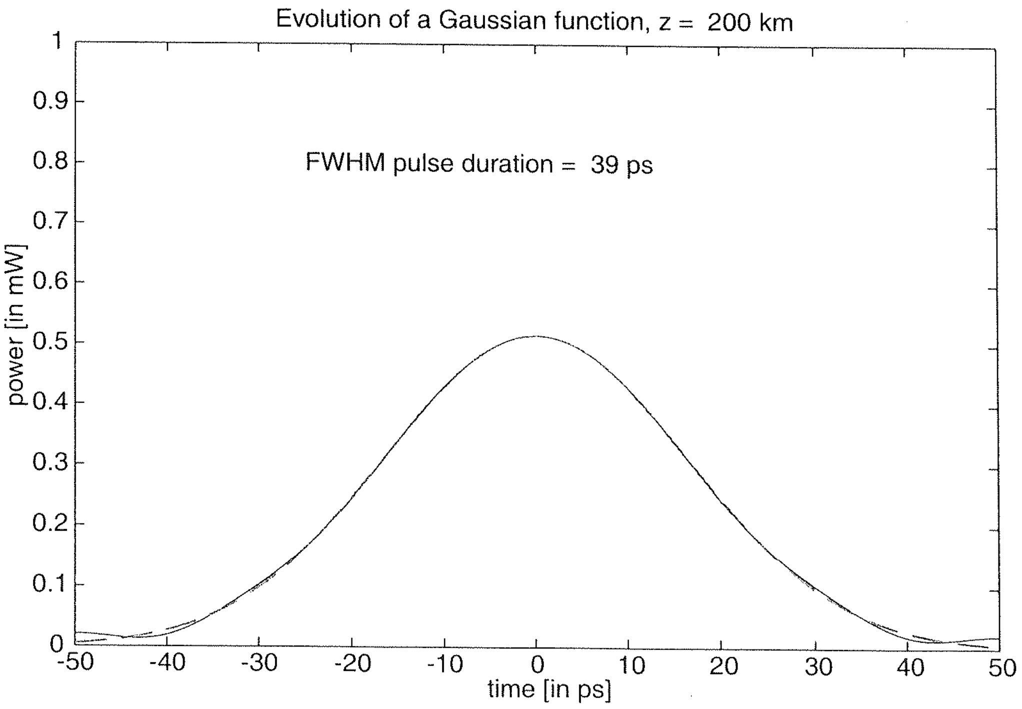
MATLAB output z = 100 km



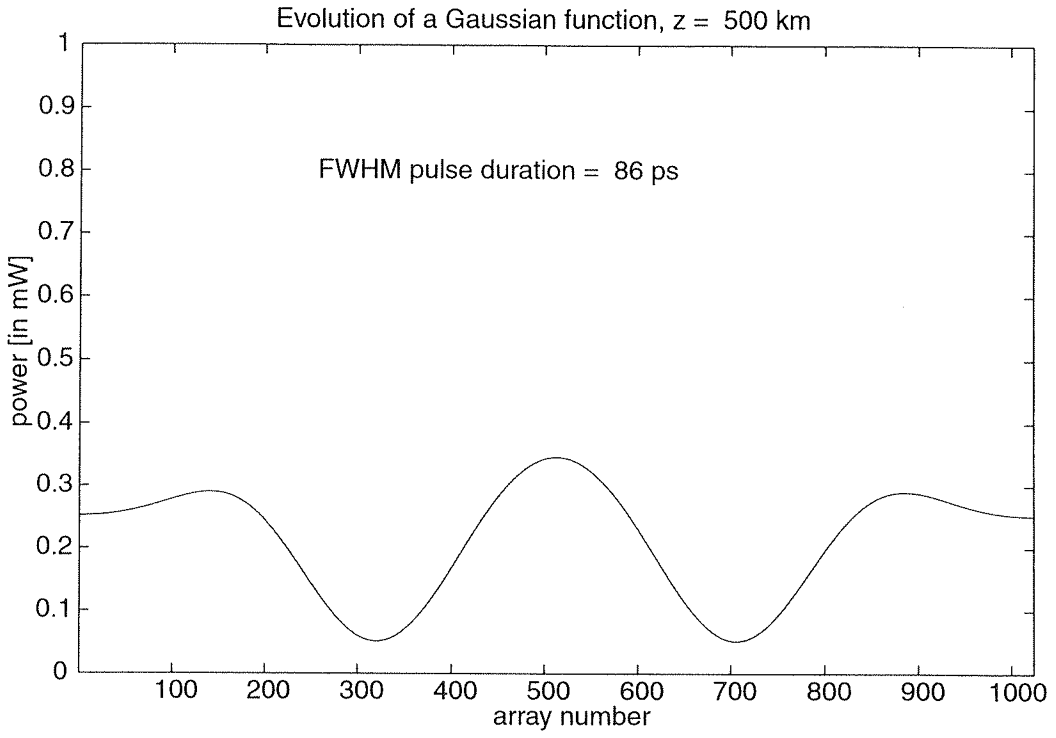


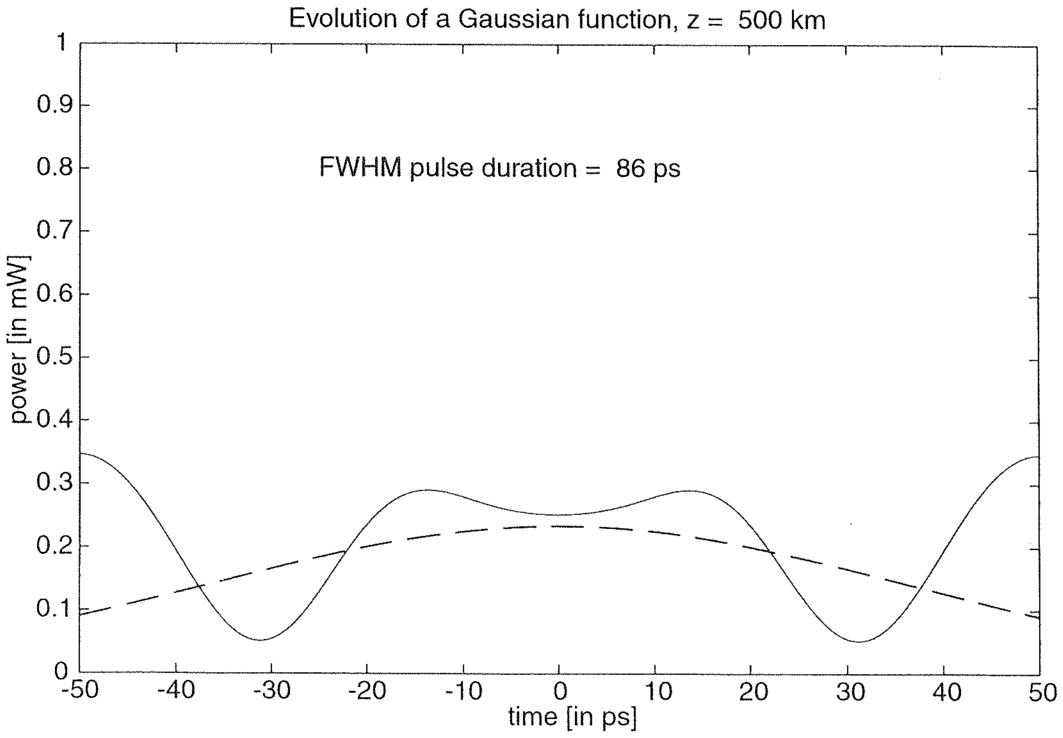
MATLAB output z = 200 km





MATLAB output *z* = 500 km





E. Demo 2A: Error vs. Operation Count

1. Higher order does not necessarily mean higher accuracy for a given amount of computation time. The reason is that it is more effort to take each step. Hence, for a given amount of computation time, one must take larger steps in the higher-order method. However, because the error diminishes faster for the higher-order method, they eventually win. Thus, the key question when picking the order to use is: "How much error is good enough?"

2. To illustrate this concept, we look at the accumulated phase error for both the split-step and Runge-Kutta methods, applied to the harmonic oscillator. In this particular case, it can be evaluated analytically. There is no point in looking at the radial error — it is zero for split-step!

3. For a fixed number of cycles in the harmonic oscillators, the number of split-step function (derivative) evaluations is 1

This is a usual measure of the computational effort. See Gear

Hence $N_{eval} = N_{steps} = 2\pi N_{cycle}/\Delta$

For Runge-Kutta, thus number is 4

Hence, $N_{eval} = 4 . N_{steps} = 8\pi N_{cycle}/\Delta$

Because $N_{eval} \propto 1/\Delta$, we define a Work function, $\Delta_{ss}^{-1} = \text{Work}$

For the same amount of Work $\boxed{\Delta_{RK} = 4\Delta_{ss}}$

In other words $\Delta_{ss} = W^{-1}$, $\Delta_{RK} = 4W^{-1}$

At, the same time, for a given N_{cycle}, N_{RK} (the number of Runge-Kutta steps) $= N_{ss}/4$

4. We now write:

a. $\text{Error}_{ss}/N_{ss} = \left| \tan^{-1}\left[\frac{\Delta_{ss}(1-\Delta_{ss}^2/4)^{1/2}}{(1-\Delta_{ss}^2/2)}\right] - \Delta_{ss}\right|$

= error/step.

$$= \left| \tan^{-1}\left[\frac{W^{-1}(1-W^{-2}/4)}{(1-W^{-2}/2)}\right] - W^{-1}\right|$$

6. $\text{Error}_{RK} / N_{SS} = \frac{1}{4}[\text{error/step}]$

$$= \frac{1}{4}\left| \tan^{-1}\left[\frac{(\Delta_{RK} - \Delta^3_{RK}/6)}{(1-\Delta^2_{RK}/2 + \Delta^4_{RK}/24)}\right] - \Delta_{RK}\right|$$

$$= \frac{1}{4}\left| \tan^{-1}\left[\frac{(4W^{-1} - 4^3 W^{-3}/6)}{(1-8W^{-2} + 4^4 W^{-4}/24)} - 4W^{-1}\right|$$

In both cases, the actual error and actual work is obtained by multiplying both axes by N_{SS}

5. In this case, Runge-Kutta rapidly becomes advantageous if the phase deviation is important.

Example: Suppose we want an error of 10^{-2} and we want to propagate 10 cycles. $\Rightarrow N_{SS} = \frac{20\pi}{\Delta} \simeq \frac{63}{\Delta}$

When $\Delta_{SS} = 0.1$, Error/step $= 4 \times 10^{-5}$ and total error $= .025$
When $\Delta_{SS} = 0.06$, Error/step $\cong 10^{-5}$, $N_{SS} \simeq 1000$ and we reach our error criterion.
With the same work, R-K's error is 2×10^{-3} (1/5 as big).

Conversely, however, at the same error, when $\Delta_{SS} = 0.09$ for R-K; so, there is a 50% savings in work. Not large

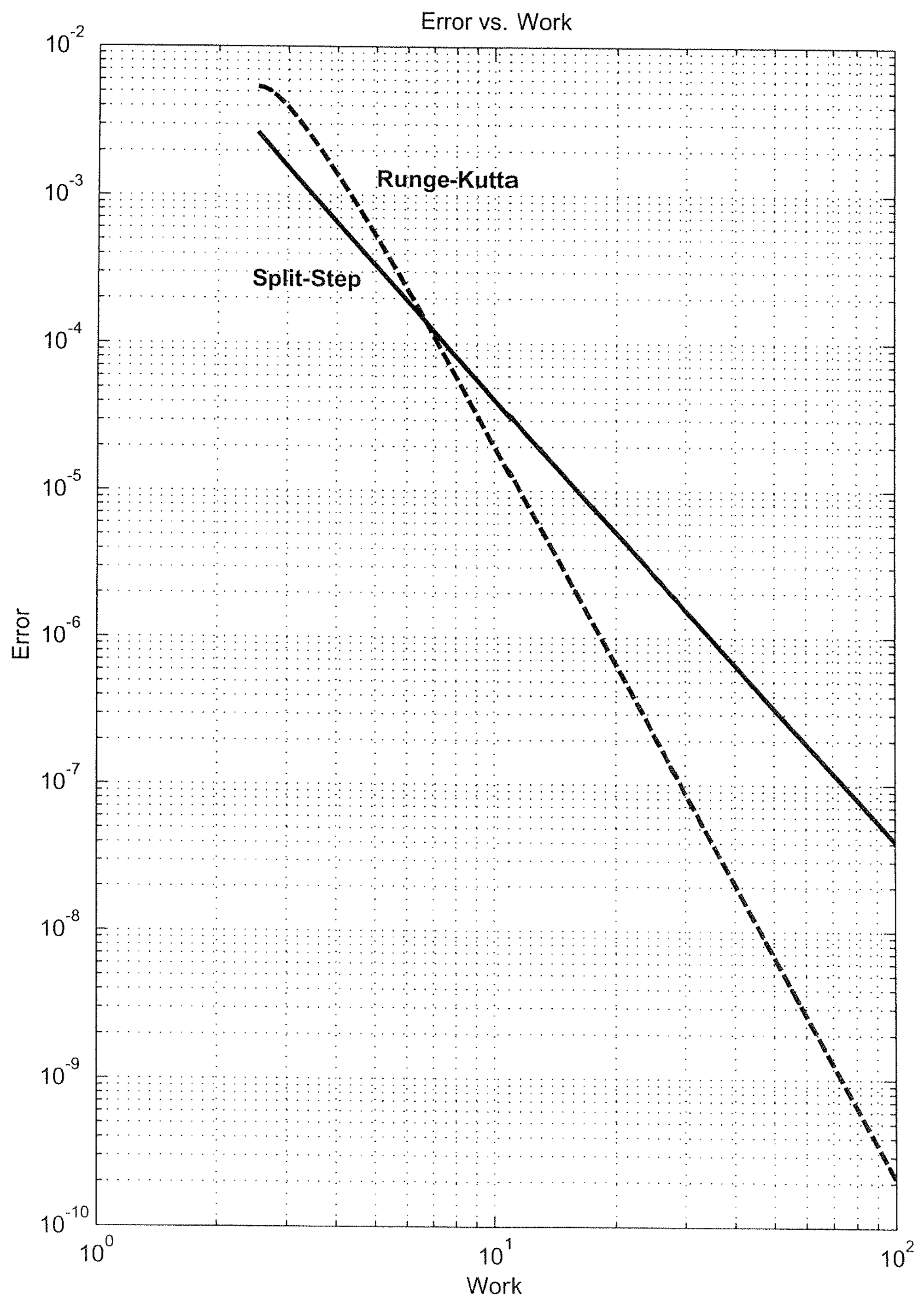
For the errors we need, split-step is adequate

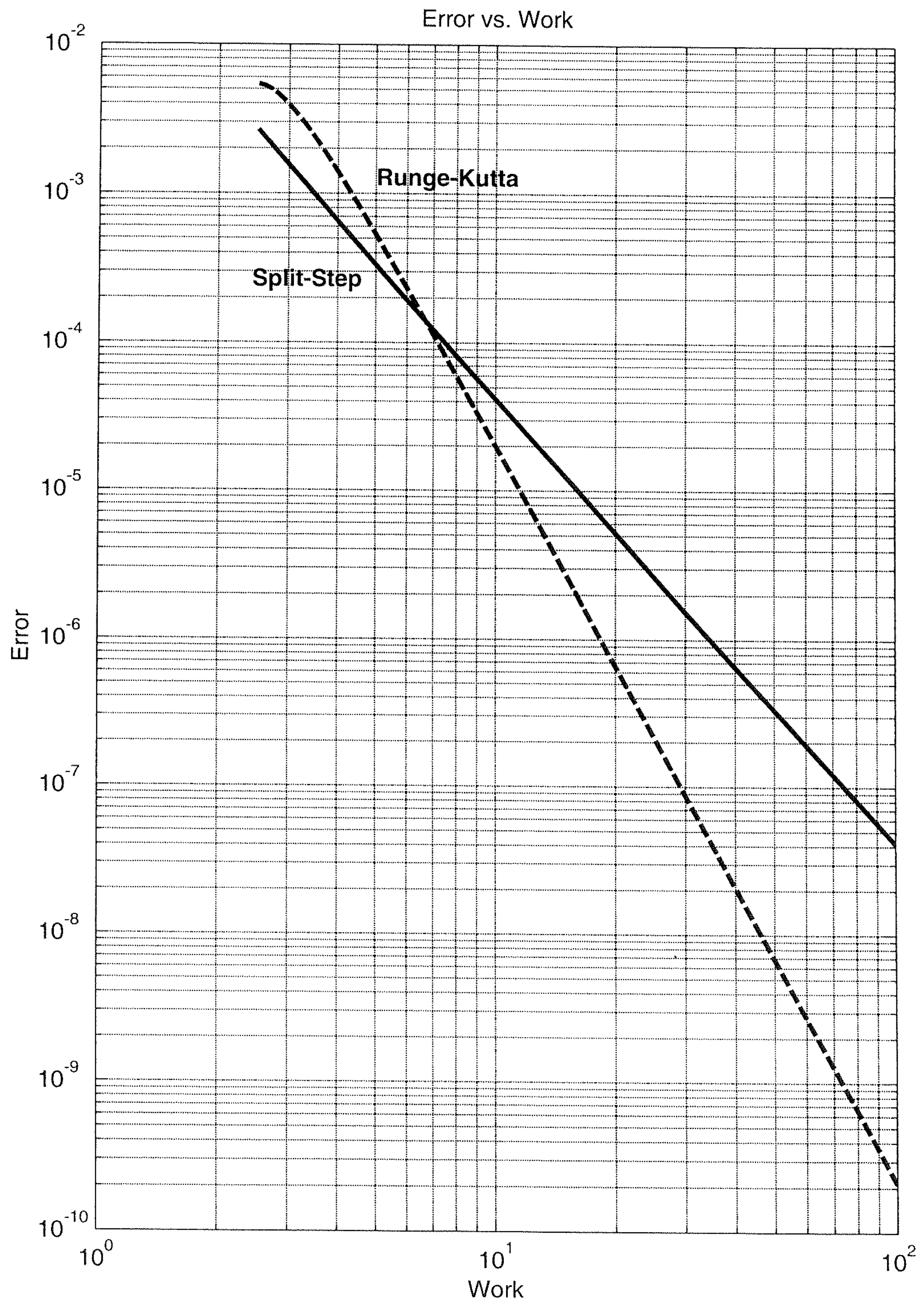
6. In reality, errors are hard to estimate. Always check your results by halving step sizes, grid sizes, etc!

MATLAB code for evaluation of phase errors vs. work for the split-step and Runge-Kutta methods applied to a harmonic oscillator (January 29, 2003 — C. R. Menyuk)

```
% Error_Plot
%
% The phase error for a fixed amount of work is plotted for both the
% split step method and the Runge-Kutta method, applied to the harmonic
% oscillator.

% The basic calculation assumes that since the operator count includes
% one function evaluation per step for the split step method, while it
% includes four function evaluations per step for the Runge-Kutta method,
% the step size for a given amount of work is four times as high for
% the latter method.  It is also inversely proportional to the step size,
% hence we set it equal to the inverse step size for the split step method
% and four times as high for the Runge-Kutta method.

% Input parameters
range = [.4 2];     % the logarithm base 10 of the range of Work
npoints = 101;     % the number of evaluation points

% Set up basic arrays
Inc = (range(2) - range(1))/(npoints - 1);  % increment of the array Work
Work = range(1):Inc:range(2);               % Set up Work array
Work = 10.^Work;
Delta2 = 1.0./Work;                         % Step sizes for split-step
Delta4 = 4.0*Delta2;                        % Step sizes for Runge-Kutta

% Calculate error arrays
err2 = atan(Delta2.*sqrt(1-Delta2.^2/4)./(1-Delta2.^2/2)) - Delta2;
err2 = abs(err2);                           % split step error
err4 = atan(Delta4.*(1-Delta4.^2/6)./(1-(Delta4.^2/2).*(1-Delta4.^2/12))) ...
    - Delta4;      err4 = 0.25*abs(err4);   % Runge-Kutta error

% Plot results on a log-log plot
loglog(Work,[err2;err4])
grid on
```

Output of MATLAB Error Determination Code




Error vs. Work
Error
Work
Runge-Kutta
Split-Step
10^-2
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8
10^-9
10^-10
10^0
10^1
10^2