0.0.1 Strongly non-paraxial regime: Poisson‘s bright spot

Summary:

- This exercise shows how a 1-D implementation of the FFT-BPM can be easily modified for two transverse dimensions...
- ... and illustrates the FFT-BPM strength in an application that bridges different proagation regimes, from strongly non-paraxial to paraxial.
- In this context, the computational complexity becomes an important issue and we touch upon question of performance and parallelization.

The physical context chosen for this work-package is that of diffraction of a collimated beam on a circular, completely opaque screen. This gives rise to the famous effect called Poisson‘s bright spot, or spot of Arago. A nice paper by R. Lucke (Eur. J. Phys. 27 (2006) 193) and related comment by G.S. Smith (Eur. J. Phys. 27 (2006) L21L23) are convenient sources to read on the mathematical background and in particular about two versions of diffraction formulas, Rayleigh-Sommerfeld and Fresnel-Kirchhoff integrals.

Form the numerical point of view, the calculation of the diffracted field in the space beyond the circular obstacle, and especially in the close vicinity of the optical axis and close to the screen, presents a good opportunity to realize how important are the non-physical parameters underlying simulations, such as the grid resolution. The reader will quickly appreciate that a calculation of the very same quantity may require very different numerical efforts depending on the choice of the simulated problem geometry.

Task 1: Simulation implementation
Possibly starting from the source developed for the previous exercise, write a program to calculate the diffracted field behind the circular obstacle when illuminated by a finite-diameter collimated beam. This will require an gextension of the FFT-BPM to two transverse dimensions.

Solution

The extension of the code from a previous exercise is not difficult. Let us comment on couple of points where mistakes occur often. As in the case with a single trnasverse dimension, one needs to prepare the linear propagator which in turn requires to calculate first the “allowable” spatial wavenumbers. These are of course obtained the same way as in one dimension. However, the realization of the propagator must be specifically two-dimensional because of the *non-paraxial* difraction regime that needs to be properly captured (recall that while paraxial regime produces propagating beam which is a direct product of two one-dimensional beams diffracting in their respective dimensions, such a reduction does not occur in general). For simplicity, it is assumed that the size of the computational box is the same along both transverse axes which we take as x and y, both sampled with NX grid points. Then in C language, the core part of the propagator implementation could be written as in this example:

Listing 1: Non-paraxial BPM propagator implementation in C

```
1
2   /* calculate transverse wavenumbers */
```

```
3    double kx[NX];
4    for(x=0;x<NX/2;x++) kx[x] = dk*x;
5    for(    ;x<NX;  x++) kx[x] = dk*(x-NX);
6
7    /* 2D propagator holder, use FFTW memory allocation */
8    fftw_complex* pxy;
9    pxy = fftw_malloc(NX*NX*sizeof(fftw_complex));
10
11   for(x=0;x<NX;x++) {
12     for(y=0;y<NX;y++) {
13       // paraxial: comment out for this exercise!
14       pxy[x+NX*y] = cexp(-I*(kx[x]*kx[x] + kx[y]*kx[y])/(2*k0)*dz )/(NX*NX);
15
16       // non-paraxial: use this version here!
17       pxy[x+NX*y]  = cexp(+I*(csqrt(k0*k0 - kx[x]*kx[x] - kx[y]*kx[y]));
18       pxy[x+NX*y] *= cexp(-I*k0*dz )/(NX*NX);
19       }
20   }
```

In the last expression, the reference carrier-wave phase is taken out. This is mostly inconsequential in the BPM context, since one is rarely interested in the absolute beam phase. Also note that when one uses a fast Fourier transform library, as FFTW in this example, it is better for performance to use the corresponding routines to allocate the propagator memory.

The physical nature of the simulated problem requires that the spectral beam propagation method is applied as a series of shorter integration steps, and at least a poor man's absorbing boundary guard is applied periodically to the solution. The reason for this is related to the occurrence of the complete spectrum of transverse wavenumbers, from very small ones, through waves that propagate almost perpendicularly to the axis, to even evanescent waves. One roughly say that each '"bundle" of plane waves that constitute a cone gives rise to the amplitude profile at certain distance from the screen. Aiming at reconstructing the latter from zero to "infinity," one must include waves with steep propagation angles, and the wavepacket that thse constitute reach the computational boundary very quickly. This is where they must be continually (or sufficiently often) destroyed by the boundary guard.

As a side note, the above also works in the opposite direction. If one only needs to claculate the beam profile at a specific large distance from the screen, a paraxial propagator will suffice as long as it carries the waves with proagation angles subtended by the axis and a line that connects the observation point on axis with the edge of the screen. In fact, this is exactly what was done in the other practical exercise with the Poisson's bright spot.

Going back to the boundary guard implementation, keep in mind that there is no universal recipe to set this up. The following code snippet is just such an example:

Listing 2: Simple boundary guard

```
1  /* 2D boundary guard holder, allocated with FFTW routine */
2  double* bxy;
3  bxy = fftw_malloc(NX*NX*sizeof(double));
4  for(x=0;x<NX;x++) {
```

```
5     for(y=0;y<NX;y++) {
6       // super-gaussian shape example: experiment with parameters!
7       bxy[x+NX*y] = exp(-pow((cx[x]*cx[x] + cx[y]*cx[y])/(LX*LX/5.0),8.0) );
8       }
9   }
```

It is to be emphasized that suitable parameters that control the shape of the guard should be obtained with experimentation and careful testing. The profile of the guard must not be too steep to minimize wave reflection from it. Moreover, a too frequent application of even a gradual absorption profile eventually creates an effectively smaller computational domain with a reflective boundary, thus defeating it own purpose. This problem is a good place to experiment with these simple absorbing boundary conditions. The reader will surely find parameter setting that will work well and many more that will not work at all. The take-home lesson should be that of extreme caution in working with boundary guards of this simplest kind.

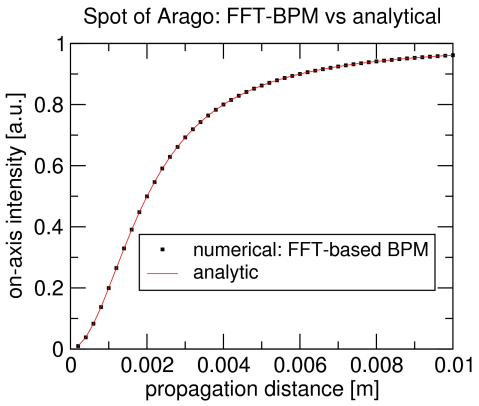
Finally, the main loop of the code is pretty much the same as in one dimension, alternating Fourier transform (in 2D), multiplication by the above propagator, following by the inverse Fourier transform. Periodically, this step should be followed by annihilating the outgoing waves that near the computational domain boundary.

The reader can find an instructors solution example in the corresponding practical exercise folder. Both single-thread and parallelized version in C and Matlab are provided as examples.

Task 2: On-axis intensity of the Poisson‘s bright spot
Reproduce in simulation the analytical expression for the on-axis intensity of the Poisson‘s bright spot. With a standing for the radius of a circular obstacle perpendicular to the beam propagation axis, the amplitude at the propagation distance z is

$$U(z) = \frac{z}{\sqrt{a^2 + z^2}} \ . \tag{1}$$

The simulation should also capture the immediate vicinity of the opaque screen, which will need the application of the non-paraxial propagator. It is left up to the reader to choose concrete parameters. The main goal is to obtain data which will show an agreement with the analytic target at least as good as that shown in the figure:



Analytic and simulated profile of the on-axis intensity vs propagation distance. Beam with a supper-Gaussian cross-section intensity profile diffracts around a circular opaque obstacle and gives rise to the famous Poisson‘s bright spot (the picture shows its intensity). The agreement between analytic and simulated solutions is essentially perfect, but this is only achievable with a reasonable numerical effort when the parameters of the problem are chosen appropriately (see text).

Solution:
At this point, I strongly encourage the reader to stop reading, and attempt the solution. One should quickly realize that it is not an easy simulation. Insuficient grid resolution may be suspect, but it can happen that even grids with several thousand points across will not produce a satisfactory agreement with the analytic formula... The deviations will mainly occur in the vicinity of the screen, while a better agrement may appear further down the axis.

Clearly, the non-paraxial nature of the optical field just beyond the screen is difficult to capture numerically, and is limitted by the maximal wavenumbers that are resolved by the numerical propagator. Points on the axis that are close to the screen are iluinated by waves that propagate essentialy perpendicularly to the beam propagation direction as a whole. Consequently, the transverse resolution of the numerical grid must be better than the wavelength of the beam.

However, because the problem states that the parameters of the simulated geometry can be freely chosen, we can cheat and by-pass the above described difficulties. Note that the wavelength does not explicity appear in the formula (1)! If for a given radius a the wavelength is so large that a/λ is a modest number, the simulation becomes very easy. This is indeed how the results shown in the figure were obtained. With $a = 2$ mm, and $\lambda = 100\mu$m, a grid of 4096×4096 point turns out to be sufficient. The solution was sampled in 50 steps with a boundary guard applied after each FFT-BPM step.

Instructor's examples provided in the practice folder in files *cmp.c* and *instructors.c* for serial and parallel implmentation in C, and in *p1.m* for a Matlab version.

This is the first practical problem in this course in which the compute time becomes an issue, and wishes that the simulation would run just a bit faster. So it is a good opportunity to experiment with parallelization, too. While Matlab can parallelize for free, and possible even without users knowledge, certain amount of work investment is required in a compiled language. The example given in this practice package is for the OpenMP, pragma-based loop parallelization.

Task 3:
How sharp is the pattern in this diffraction scenario is, depends on how sharp and smooth the boundary of the obstacle screen is. Design a modification of the initial condition that mimics a fuzzy obstacle edge, and show that when the fuzziness affects one Fresnel zone, the central part of the pattern starts to degrade.