DOING PHYSICS WITH MATLAB

Ian Cooper

matlabvisualphysics@gmail.com

A COMPUTATIONAL APPROACH TO ELECTROMAGNETIC THEORY

[2D] NUMERICAL INTEGRATION POTENTIAL FROM A SQUARE PLATE

DOWNLOAD DIRECTORIES FOR MATLAB SCRIPTS

Google drive

GitHub

cemb001.m

Calculation of the electrostatic potential along the symmetry axis of a uniformly charged square plate

simpson2d.m

[2D] integral

In electromagnetism it is often necessary to evaluate a **surface integral**. The integration can often be done simply by a numerical approximation using the function **simpson2d.m**.

However, when using numerical tools, one must keep in mind that they never give an exact answer. The accuracy of the numerical result usually depends upon the number of grid points used in the evaluation of the integral and on how well the geometry of the problem is represented by a finite mesh. If the method works correctly, the computed answer should converge towards the exact result as the number of grid points increases (increased **resolution**). For any numerical computation, one should test the **convergence** and if possible, compare the result to an exact analytical answer.

We will use a simple example, namely the calculation of the **electrostatic potential** $\phi(0,0,z)$ **along the symmetry axis of a uniformly charged square plate**. The square is the region

$$-L < x < L \quad -L < y < L \quad z = 0$$

and the surface charge density $\rho(x,y)$ is constant

$$\rho(x,y) = \rho_0$$

The electrostatic potential from the charged square plate is

(1) $$\phi(0,0,z) = \left(\frac{\rho_0}{4\pi\,\varepsilon_0}\right)\int_{-L}^{L}\int_{-L}^{L}\frac{dx'dy'}{\sqrt{x'^2 + y'_2 + z^2}}$$

The evaluation of the integral is done using the **simpson2d.m** function for a x-y mesh with NxN grid points where N must be an **odd** integer.

The electrostatic potential $\phi_{PS}(r)$ from a point charge *Q* at a point with displacement *r* is

(2) $$\phi_{PS}(r) = \frac{Q}{4\pi\,\varepsilon_0\; r}$$

To test convergence and the evaluation of the numerical method we can compare the predictions from equations 1 and 2. Far from the plate, the potential from the plate should converge to the value from the point charge. We can also increment N to find the its minimum value for convergence.

The results of the modelling using the Script **cemB001.m** are displayed in the figures 1 and 2.

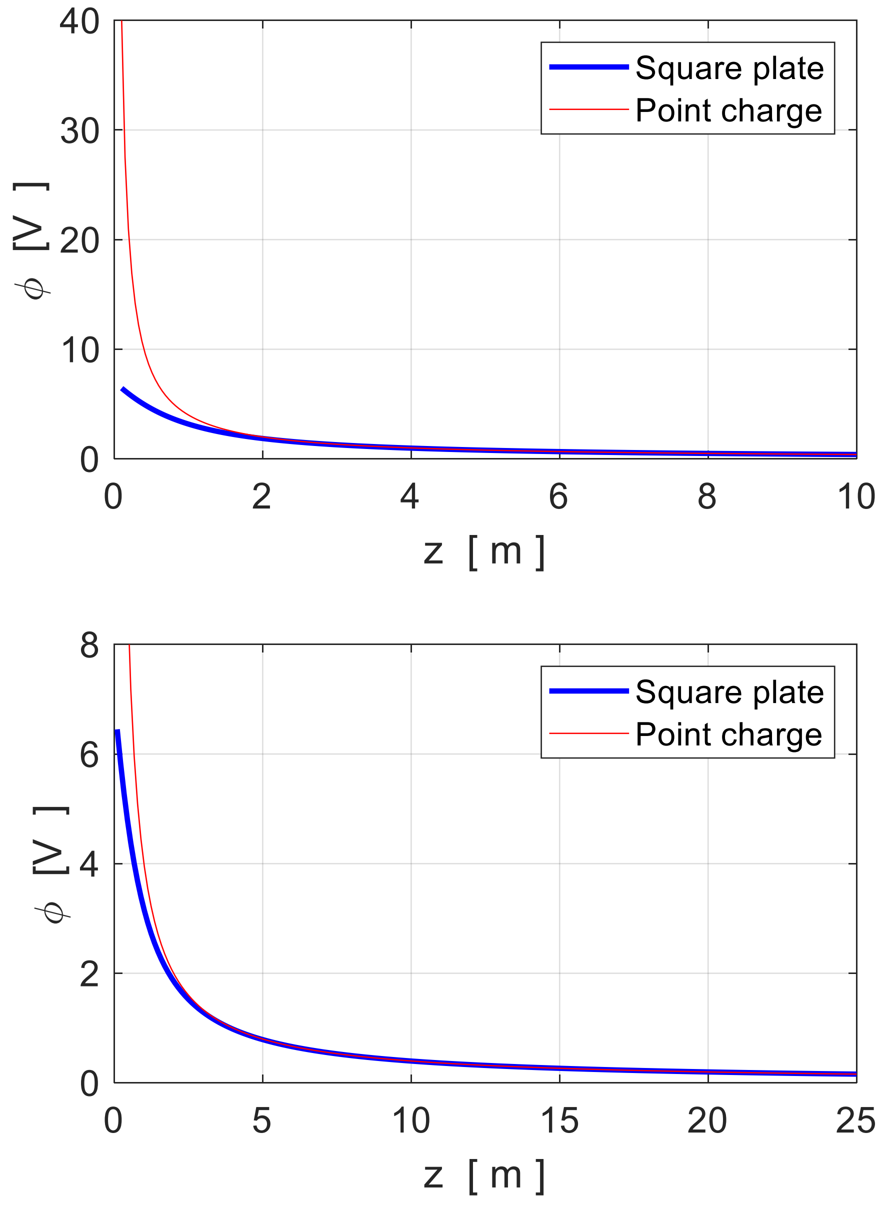

Fig. 1. Even at distances for $z > \sim 2$ m the plate appears to be a point source and there is excellent agreement between the predictions of equations 1 and 2.

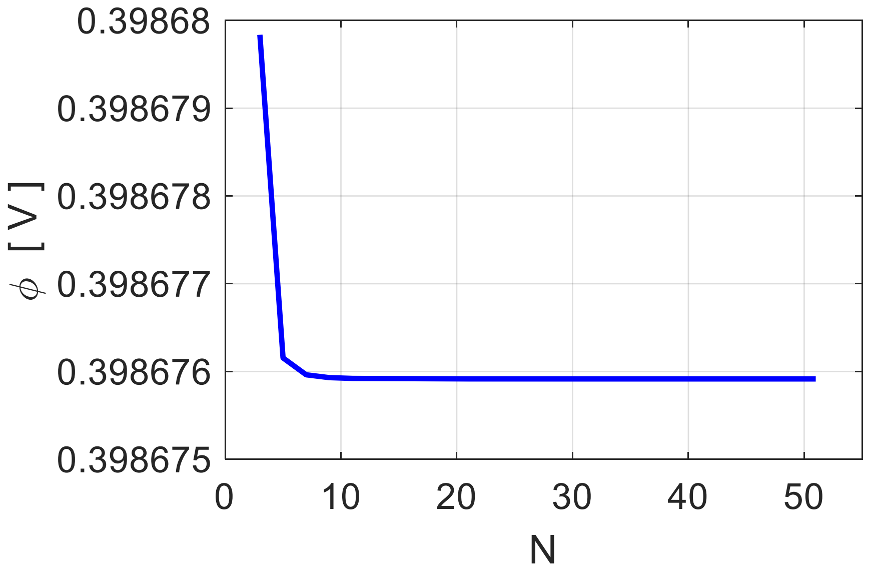

Fig. 2. Even with a small value for N (N > 9) the computation converges to a fixed value.

The [2D] form of Simpson’s rule in general is a good numerical method to evaluate [2D] surface integrals.

As N increases, it does not always guarantee convergence. In some instances, the value of the integral may oscillate.

cemB001.m

Section of the Script that performs the computations

```
% Length of square plate 2*L
  L = 1;
% Permitty of free space / Coulomb constant
  eps0 = 8.85e-12;
  kC = 1/(4*pi*eps0);
% Surface charge density of plate
%  rho0 = 1;
  rho0 = 1/kC;
% Constant / Charge
  K = rho0/(4*pi*eps0);
  Q = (2*L)^2*rho0;
% X Y Z grids
  N = 219;   % number of grid pints: must be an ODD number
% Input z range >>>>>>
  zMin = 0.1*L; zMax = 25*L;
  xMin = -L; xMax = L;
  yMin = -L; yMax = L;
  z = linspace(zMin,zMax,N);
  x = linspace(xMin,xMax,N);
  y = linspace(yMin,yMax,N);
 [xx, yy] = meshgrid(x,y);
% Function to be integrated
  pot = zeros(N,1);
  for n = 1:N
    fn = 1./(sqrt(xx.^2 + yy.^2 + z(n).^2));
    pot(n) = simpson2d(fn,xMin,xMax,yMin,yMax);
  end
% Potential
  pot = K.*pot;
% Potential from a point source
  potPS = Q./((4*pi*eps0).*z);
% Convergence
=========================================================
  zC = 10;
  potPSC = Q./((4*pi*eps0).*zC);
  n = [3 5 7 9 11 21 31 41 51]';
  potC = zeros(length(n),1);
  for c = 1 : length(n)
    x = linspace(xMin,xMax,n(c));
    y = linspace(yMin,yMax,n(c));
    [xx, yy] = meshgrid(x,y);
    fnC = 1./(sqrt(xx.^2 + yy.^2 + zC.^2));
   potC(c) = simpson2d(fnC,xMin,xMax,yMin,yMax);
  end
   potC = K*potC;
```