```
> restart;
  with(PDEtools):
  with(LinearAlgebra):
  with(plots):
  interface(rtablesize=20):
```

Finite difference solution of Poisson's equation

The purpose of the worksheet is to solve Poisson's equation using finite differencing. Poisson's equation is:

```
> poisson_eq := diff(u(x,y),x,x) + diff(u(x,y),y,y) = f(x,y);
```

$$poisson\_eq := \frac{\partial^2}{\partial x^2} u(x, y) + \frac{\partial^2}{\partial y^2} u(x, y) = f(x, y) \quad (1)$$

We assume Dirichlet boundary conditions of the form

```
> BCs := [u(x,y_bottom)=0,u(x,y_top)=0,u(x_left,y)=0,u(x_right,y)=
  0];
```

$$BCs := [u(x, y\_bottom) = 0, u(x, y\_top) = 0, u(x\_left, y) = 0, u(x\_right, y) = 0] \quad (2)$$

We will be using the fivepoint stencil of the PDE. This obtained with the following code:

```
> centered_stencil := proc(r,N,{direction := x})
      local n, stencil, vars, beta_sol:
      n := floor(N/2):
      if (direction = y) then:
         stencil := D[2$r](u)(x,y) - add(beta[i]*u(x,y+i*h),i=-n..
  n);
         vars := [u(x,y),seq(D[2$i](u)(x,y),i=1..N-1)];
      else:
         stencil := D[1$r](u)(x,y) - add(beta[i]*u(x+i*h,y),i=-n..
  n);
         vars := [u(x,y),seq(D[1$i](u)(x,y),i=1..N-1)];
      fi:
      beta_sol := solve([coeffs(collect(convert(series(stencil,h,
  N),polynom),vars,'distributed'),vars)]):
      stencil := subs(beta_sol,stencil);
      convert(stencil = convert(series(stencil,h,N+2),polynom),
  diff);
  end proc:
  x_stencil := isolate(lhs(centered_stencil(2,3,direction=x)),diff
  (u(x,y),x,x));
  y_stencil := isolate(lhs(centered_stencil(2,3,direction=y)),diff
  (u(x,y),y,y));
  poisson_stencil := expand(subs(x_stencil,y_stencil,poisson_eq*
  h^2));
```

$$x\_stencil := \frac{\partial^2}{\partial x^2} u(x, y) = \frac{u(x-h, y)}{h^2} - \frac{2\,u(x, y)}{h^2} + \frac{u(x+h, y)}{h^2}$$

$$y\_stencil := \frac{\partial^2}{\partial y^2} u(x, y) = \frac{u(x, y-h)}{h^2} - \frac{2\,u(x, y)}{h^2} + \frac{u(x, y+h)}{h^2}$$

$$poisson\_stencil := u(x-h, y) - 4\,u(x, y) + u(x+h, y) + u(x, y-h) + u(x, y+h) = h^2 f(x, y) \quad (3)$$

We re-label the u's and f(x,y) as follows:

```
> Subs := [seq(seq(u(x+jj*h,y+ii*h)=u[i+ii,j+jj],ii=-1..1),jj=-1.
  .1),f(x,y)=f[i,j]/h^2];
```

$$Subs := \left[u(x-h,y-h)=u_{i-1,j-1},\ u(x-h,y)=u_{i,j-1},\ u(x-h,y+h)=u_{i+1,j-1},\ u(x,y-h)=u_{i-1,j},\ u(x,y)=u_{i,j},\ u(x,y+h)=u_{i+1,j},\ u(x+h,y-h)=u_{i-1,j+1},\ u(x+h,y)=u_{i,j+1},\ u(x+h,y+h)=u_{i+1,j+1},\ f(x,y)=\frac{f_{i,j}}{h^2}\right] \quad (4)$$

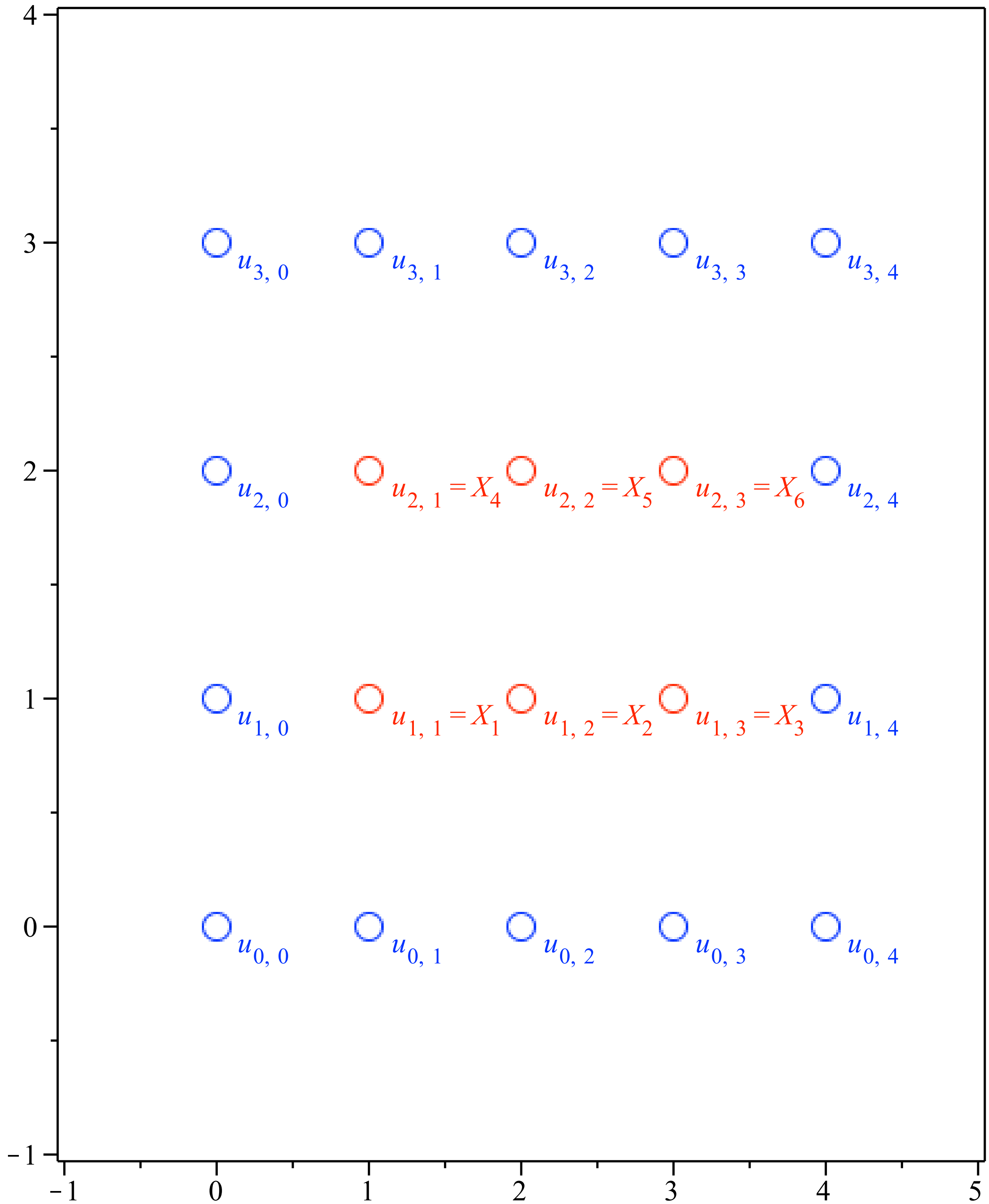
Our convention is that i increases with y and j increases with x. Putting these into our stencil we get

```
> poisson_stencil := subs(Subs,poisson_stencil);
```

$$poisson\_stencil := u_{i,j-1} - 4\,u_{i,j} + u_{i,j+1} + u_{i-1,j} + u_{i+1,j} = f_{i,j} \quad (5)$$

Let's consider the situation were we have M points in the x direction times N points in the y direction where we want to know the field values. We assume boundary conditions are given such that u is zero on the boundary. Here is a plot for a specific choice of M and N:

```
> M := 3:
  N := 2:
  p1 := plot([seq(seq([j,i],j=1..M),i=1..N)],view=[-1..M+2,-1..
  N+2],style=point,symbolsize=20,symbol=circle,axes=boxed):
  p2 := textplot([seq(seq([j+0.1,i,u[i,j]=X[(i-1)*M+j]],j=1..M),i=
  1..N)],align=[right,below],color=red):
  p3 := plot([seq([0,i],i=1..N),seq([M+1,i],i=1..N),seq([i,0],i=0..
  M+1),seq([i,N+1],i=0..M+1)],view=[-1..M+1,-1..N+1],style=point,
  symbolsize=20,symbol=circle,axes=boxed,color=blue):
  p4 := textplot([seq([0.1,i,u[i,0]],i=1..N),seq([M+1+0.1,i,u[i,
  M+1]],i=1..N),seq([i+0.1,0,u[0,i]],i=0..M+1),seq([i+0.1,N+1,u
  [N+1,i]],i=0..M+1)],align=[right,below],color=blue):
  display([p1,p2,p3,p4]);
```



We don't know u at each of the interior (red) points, but we do know u at each of the exterior (blue) points. poisson_stencil can be evaluated at each of the red points to give N*M linear equations for the N*M unknown u[i,j]'s. If we arrange he unknown u[i,j]'s into a single vector X as indicated in the plot, this is a matrix equation A.X = B. The following code generates A and B for the choices of N and M above:

```
> eq := 'eq':
  COUNT := 0:
  u := 'u':
  for i from 1 to N do:
    for j from 1 to M do:
```

```
        COUNT := COUNT + 1:
        eq[COUNT] := poisson_stencil;
        print(eq[COUNT]);
    od;
  od;
  vars := [seq(seq(u[i,j],j=1..M),i=1..N)];
  A,B := GenerateMatrix(convert(eq,list),vars);
```

$$u_{1,0} - 4\,u_{1,1} + u_{1,2} + u_{0,1} + u_{2,1} = f_{1,1}$$

$$u_{1,1} - 4\,u_{1,2} + u_{1,3} + u_{0,2} + u_{2,2} = f_{1,2}$$

$$u_{1,2} - 4\,u_{1,3} + u_{1,4} + u_{0,3} + u_{2,3} = f_{1,3}$$

$$u_{2,0} - 4\,u_{2,1} + u_{2,2} + u_{1,1} + u_{3,1} = f_{2,1}$$

$$u_{2,1} - 4\,u_{2,2} + u_{2,3} + u_{1,2} + u_{3,2} = f_{2,2}$$

$$u_{2,2} - 4\,u_{2,3} + u_{2,4} + u_{1,3} + u_{3,3} = f_{2,3}$$

$$vars := \left[u_{1,1}, u_{1,2}, u_{1,3}, u_{2,1}, u_{2,2}, u_{2,3}\right]$$

$$A, B := \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix}, \begin{bmatrix} -u_{1,0} + f_{1,1} - u_{0,1} \\ f_{1,2} - u_{0,2} \\ f_{1,3} - u_{1,4} - u_{0,3} \\ -u_{2,0} + f_{2,1} - u_{3,1} \\ -u_{3,2} + f_{2,2} \\ -u_{3,3} + f_{2,3} - u_{2,4} \end{bmatrix} \qquad \textbf{(6)}$$

The matrix A is composed of a block diagonal matrix plus a banded matrix. The vector B is made up of the boundary data as represented by the blue nodes in the above plot. After staring at this for a while (and playing around with the values of M and N), you'll see how to construct A for and N and M and how to construct B knowing the boundary data and the source function. These procedures do just that:

```
> AA := proc(N,M)
      local COUNT, C, E, F, G:
      COUNT := N*M:
      C := BandMatrix([1,-4,1],1,M):
      E := DiagonalMatrix([C$N]):
      F := BandMatrix([1,0$(2*M-1),1],M,COUNT):
      G := E + F:
  end proc:

  BB := proc(bottom,top,left,right,f)
        local N, M, q, i:
        N := Dimension(left):
        M := Dimension(bottom):
        q[1] := Vector([-bottom[1]-left[1]+f[1,1],seq(-bottom[j]+f
  [1,j],j=2..M-1),-bottom[M]-right[1]+f[1,M]]);
        for i from 2 to N-1 do:
            q[i] := Vector([-left[i]+f[i,1],seq(f[i,j],j=2..M-1),-
  right[i]+f[i,M]]):
        od:
        q[N] := Vector([-top[1]-left[N]+f[N,1],seq(-top[j]+f[N,j],j=
```

```
    2..M-1),-top[M]-right[N]+f[N,M]]);
         Vector(convert(q,list));
    end proc:
```

Notice the arguments of the BB procedure: it takes the values of the field at the bottom, top, left and right of the domain arranged into vectors and the source field values f arranged in a matrix. (At this stage, BB will work for arbitrary boundary data, later we will specialize to the Dirichlet case.) Notice that the corner nodes (u[0,0], etc) don't have to be included because the fivepoint stencil will never rference them. Hence, bottom is an M-dimensional vector, left is an N-dimensional vector, etc. We can test our procedures by generating dummy data:

```
> Bottom := Vector([seq(u[0,j],j=1..M)]):
  Top := Vector([seq(u[N+1,j],j=1..M)]):
  Left := Vector([seq(u[i,0],i=1..N)]):
  Right := Vector([seq(u[i,M+1],i=1..N)]):
  F := Matrix([seq([seq(f[i,j],j=1..M)],i=1..N)]):
  Bottom, Top, Left, Right, F;
```

$$\begin{bmatrix} u_{0,1} \\ u_{0,2} \\ u_{0,3} \end{bmatrix}, \begin{bmatrix} u_{3,1} \\ u_{3,2} \\ u_{3,3} \end{bmatrix}, \begin{bmatrix} u_{1,0} \\ u_{2,0} \end{bmatrix}, \begin{bmatrix} u_{1,4} \\ u_{2,4} \end{bmatrix}, \begin{bmatrix} f_{1,1} & f_{1,2} & f_{1,3} \\ f_{2,1} & f_{2,2} & f_{2,3} \end{bmatrix} \tag{7}$$

Then, subtracting the A and B determined above from the direct examination of the linear system from the procedure output generates zero, as required.

```
> AA(N,M)-A,BB(Bottom,Top,Left,Right,F)-B;
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

We are almost ready to write down the code to solve the Poisson equation. Our procedure will take an x-range and a y-range to define the rectangular region we are solving over. Also, we will specify the total number of nodes N_tot inside our computational domain rather than h, N or M directly. The relationship between N_tot, h, the width (delta_x), and the height (delta_y) of the solution region is given by the following:

```
> N := 'N':
  M := 'M':
  eq1 := h = delta_x/(M+1);
  eq2 := h = delta_y/(N+1);
  eq3 := N_tot = N*M;
  eq4 := h = solve(subs(isolate(eq1,M),isolate(eq2,N),eq3),h)[1];
```

$$eq1 := h = \frac{delta\_x}{M+1}$$

$$eq2 := h = \frac{delta\_y}{N+1}$$

$$eq3 := N\_tot = N\,M$$

$$eq4 := h = \frac{1}{2} \frac{1}{N\_tot - 1} \left(-delta\_y - delta\_x + \sqrt{delta\_y^2 - 2\, delta\_y\, delta\_x + delta\_x^2 + 4\, delta\_y\, delta\_x\, N\_tot} \right) \qquad (9)$$

PoissonSolve generates the solution of our problem using LinearSolve. The output is a 3D plot:

```
> PoissonSolve := proc(N_tot,f,{x:=-1..1,y:=-1..1})
        local h, Bottom, Top, Left, Right, sol, i, Data, sys,
              x_left, y_bottom, delta_x, delta_y, M, N, X, Y, F;
        x_left := lhs(x):
        y_bottom := lhs(y):
        delta_x := rhs(x)-lhs(x):  # width of computational domain
        delta_y := rhs(y)-lhs(y):  # height of computational domain
        h := evalf(1/2/(N_tot-1)*(-delta_y-
             delta_x+(delta_y^2-2*delta_y*
             delta_x+delta_x^2+4*delta_y*
             delta_x*N_tot)^(1/2))); # fix the stepsize
        M := round(delta_x/h - 1):   # number of points in x
   direction
        N := round(delta_y/h - 1):   # number of points in y
   direction
        X := j -> x_left + j*h:      # gives the x coordinate of the
   (i,j) lattice point
        Y := i -> y_bottom + i*h:    # gives the y coordinate of the
   (i,j) lattice point
        Bottom := Vector(1..M,datatype=float):  # initialize
   boundary data
        Top := Vector(1..M,datatype=float):     # initialize
   boundary data
        Left := Vector(1..N,datatype=float):    # initialize
   boundary data
        Right := Vector(1..N,datatype=float):   # initialize
   boundary data
        F := Matrix([seq([seq(f(X(j),Y(i))*h^2,
             j=1..M)],i=1..N)],datatype=float): # here is the matrix
   for the source data
        sys := AA(N,M),BB(Bottom,Top,Left,Right,F);    # define
   linear system to solve
        sol := LinearSolve(sys):       # solve the system using
   LinearSolve
        Data[0] := [seq([X(j),Y(0),0],j=0..M+1)]:      # following
   code sets data for surfplot
        for i from 1 to N do:
            Data[i] := [[X(0),Y(i),Left[i]],seq([X(j),Y(i),sol[M*
   (i-1)+j]],j=1..M),[X(M+1),Y(i),Right[i]]]:
        od:
        Data[N+1] := [seq([X(j),Y(N+1),0],j=0..M+1)]:
        Data := convert(Data,list):
        surfdata(Data,axes=boxed,labels=["x","y","u(x,y)"],shading=
   zhue,style=patchcontour); # surfplot generates the output
   end proc:
```

Here is an example problem the calculates the potential around a ring of 2*n alternating electric charges. We model the charges as discs of radius r located a distance of R away from the origin and with charge density +1 or -1. For n = 1, this is an electric dipole.:

```
> n := 2;
  r := 4;
  R := 5;
```

```
x_range := -15..15;
y_range := -15..15;
f := (x,y) -> add((-1)^i*Heaviside(r^2-(x-R*cos(Pi*i/n))^2 - (y-
R*sin(Pi*i/n))^2),i=0..2*n-1);
plot3d(f(x,y),x=x_range,y=y_range,shading=zhue,axes=boxed,style=
patchnogrid,title="Source function f(x,y)");
solution := PoissonSolve(1000,f,x=x_range,y=y_range):
```

$$n := 2$$

$$r := 4$$

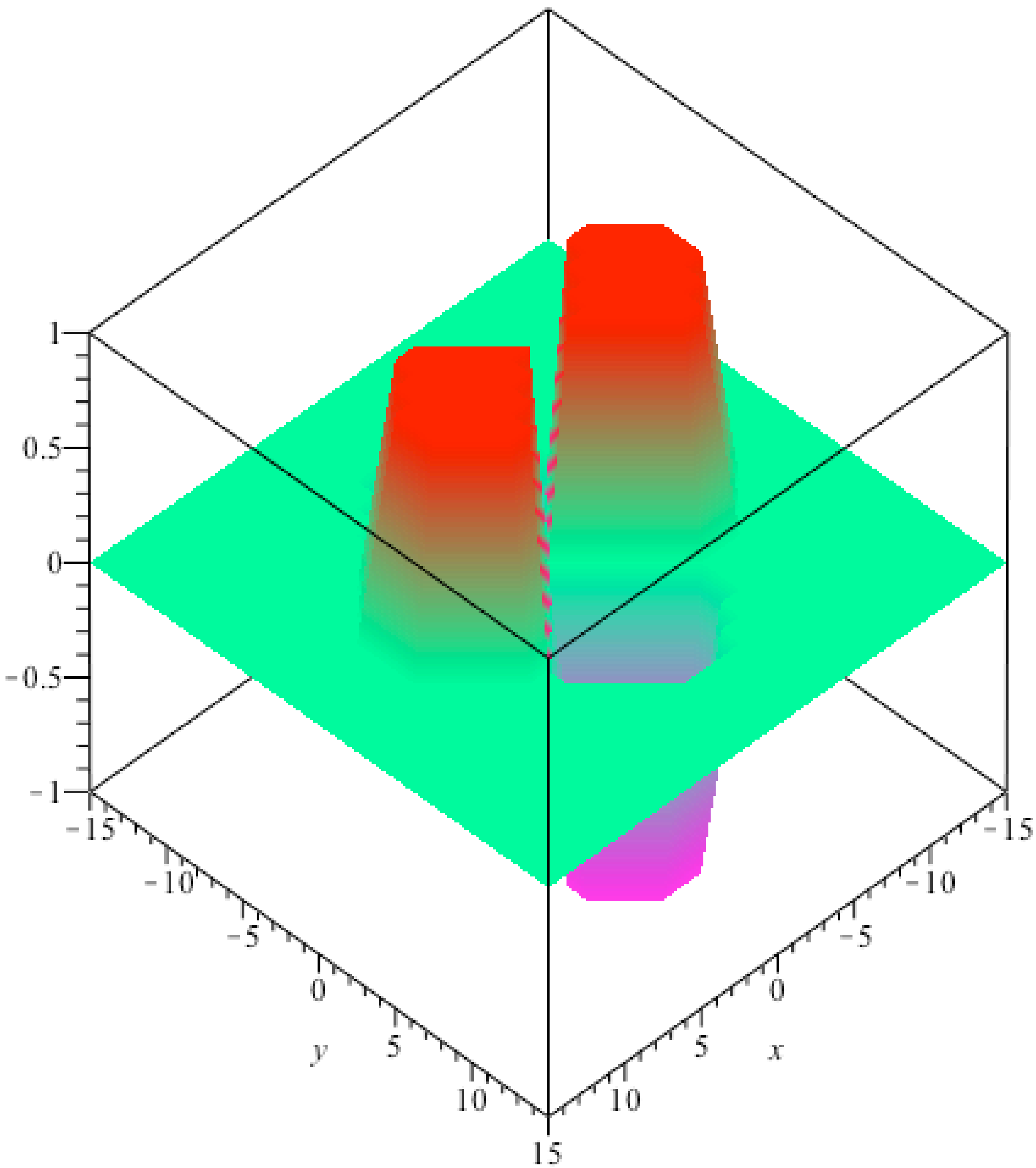
$$R := 5$$

$$x\_range := -15..15$$

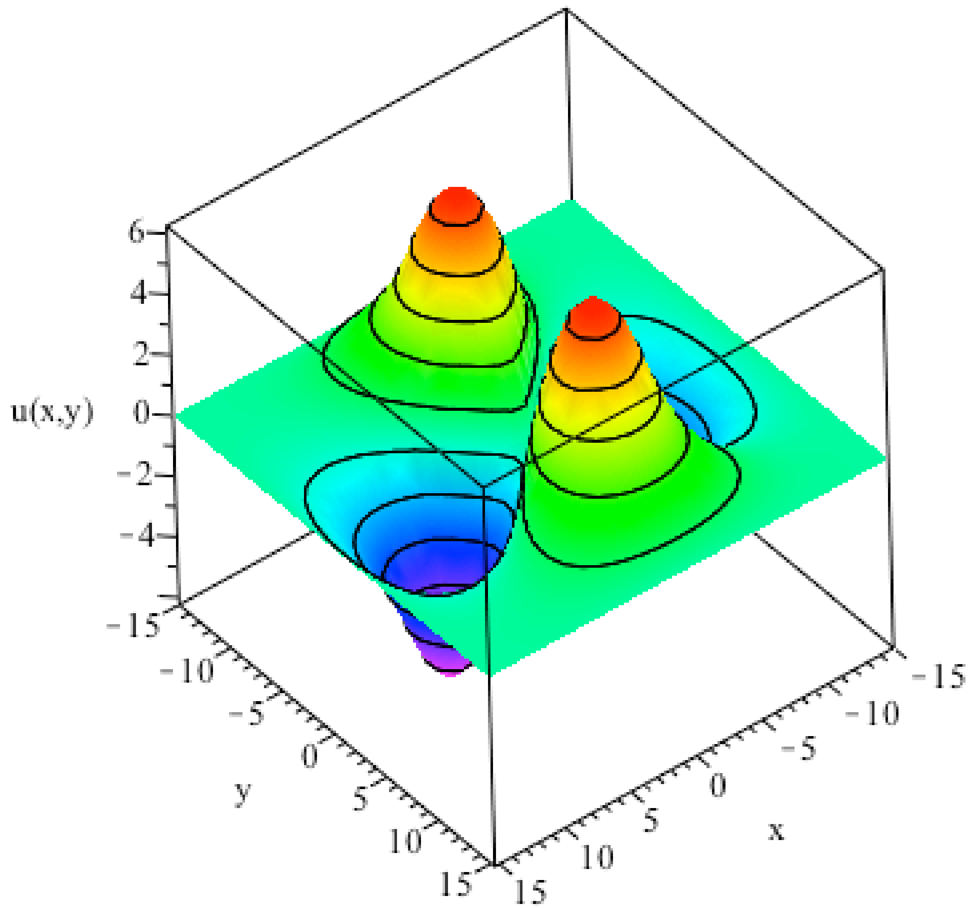
$$y\_range := -15..15$$

$$f := (x, y) \rightarrow add\left((-1)^i \text{Heaviside}\left(r^2 - \left(x - R\cos\left(\frac{\pi i}{n}\right)\right)^2 - \left(y - R\sin\left(\frac{\pi i}{n}\right)\right)^2\right), i = 0..2n - 1\right)$$

Source function f(x,y)




```
> solution;
```



u(x,y)
6
4
2
0
−2
−4
−15
−10
−5
0
y
5
10
15
15
10
5
0
x
−5
−10
−15