Scientific Computation II Final Exam

Michael Nameika

May 2023

Exercise 1: Application of RBF Interpolation - 2D reconstruction from a point cloud

See HW4.

Exercise 2: 2D and 3D reconstruction from a point cloud

(a) Modify the RBF interpolation to include a polynomial correction. In 2D, this amounts to searching the interpolant in the form

$$F(\mathbf{x}) = \sum_{j=1}^{N} c_j \phi(\|\mathbf{x} - \mathbf{x}_j\|) + p(x)$$

where $p(\mathbf{x}) = p(x, y) = \gamma_1 + \gamma_2 x + \gamma_3 y$. [See Fornberg (2016) Part 1 for details]

Implementing the polynomial correction in 2D, our problem becomes one of solving for $M + 3$ coefficients, the M coefficients as in exercise 1 plus γ_1, γ_2, and γ_3. We also require

$$\sum_{i=1}^{M} c_i = \sum_{i=1}^{M} c_i x_i = \sum_{i=1}^{M} c_i y_i = 0.$$

And so our linear system then becomes

$$\left[\begin{array}{ccc|ccc} & & & 1 & x_1 & y_1 \\ & \phi(\|x_i - x_j\|) & & \vdots & \vdots & \vdots \\ & & & 1 & x_M & y_M \\ \hline 1 & \cdots & 1 & 0 & 0 & 0 \\ x_1 & \cdots & x_M & 0 & 0 & 0 \\ y_1 & \cdots & y_M & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} c_1 \\ \vdots \\ c_M \\ \hline \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{array}\right] = \left[\begin{array}{c} 0 \\ \alpha \\ -\alpha \\ \hline 0 \\ 0 \\ 0 \end{array}\right]$$

Modifying the script from exercise 1, we find the following plots using the polynomial correction:

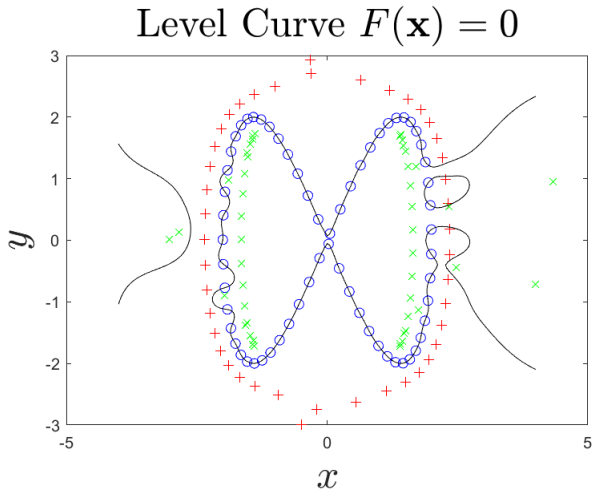


Figure 1: Infinity Lissajous point cloud with polynomial correction ($\alpha = 0.1$)

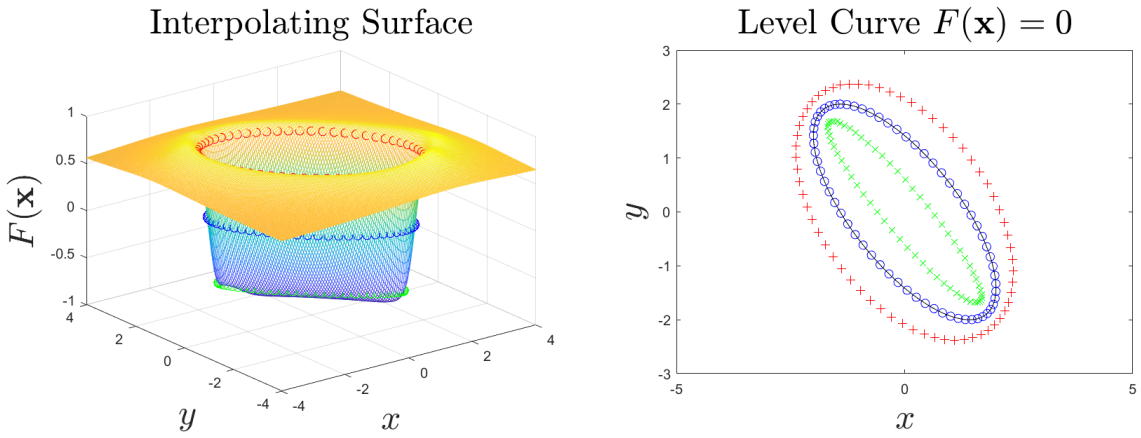


Figure 2: "Skewed" ellipse point cloud with polynomial correction ($\alpha = 0.7$)

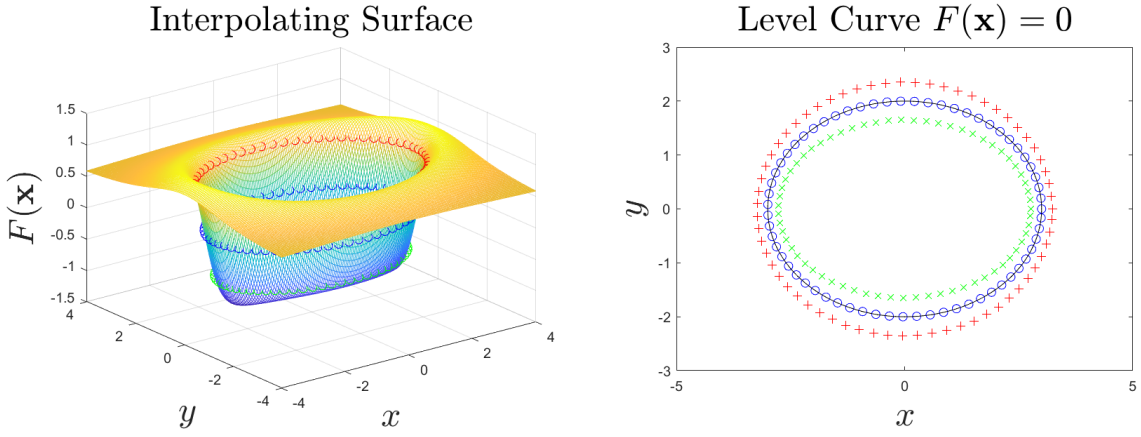


Figure 3: Ellipse point cloud with polynomial correction ($\alpha = 0.4$)

(b) Solve the analogous problem in 3D, stating that given a 3D point cloud, find an implicit surface that approximates these N points using RBF interpolants.

Modifying the script from exercise 1 for a 3D implementation, we change the computation of the outward normal at a point x_j to include any point within a predefined radius around x_j and using the built-in `isosurface` command in `MATLAB` to plot a level surface, we find the following plots for an interpolation of a spherical point cloud of radius one in 3D (the spherical point cloud was generated by a parameterization of a sphere. See the attached `my3DPointCloudRBF.m` script for additional details.):

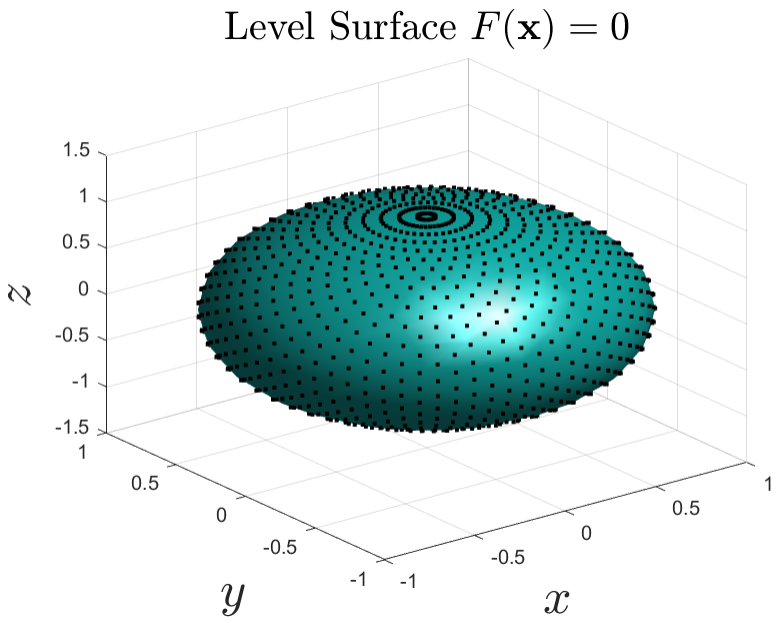


Figure 4: Level surface $F(x) = 0$ of interpolating function utilizing the IMQ RBF. Black dots are of the x^0 point cloud.

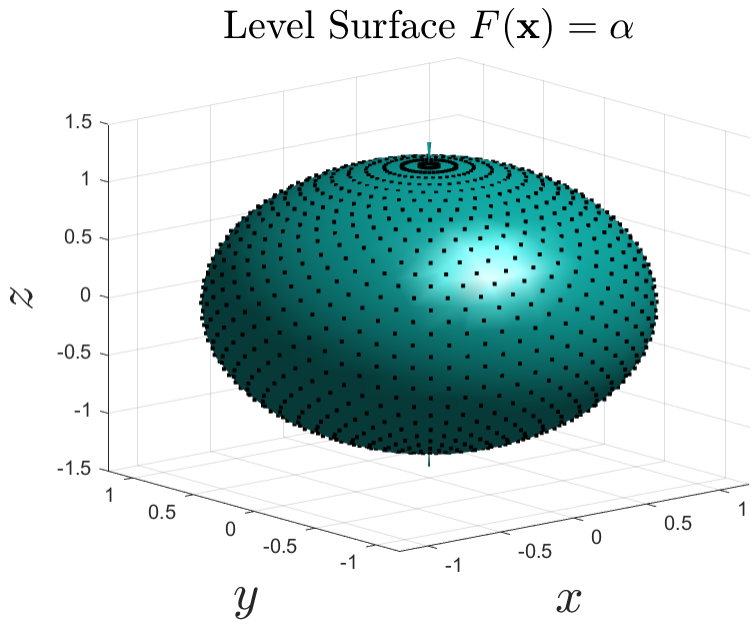


Figure 5: Level surface $F(x) = \alpha$ of interpolating function utilizing IMQ RBFs. Black dots are of the x^+ point cloud.

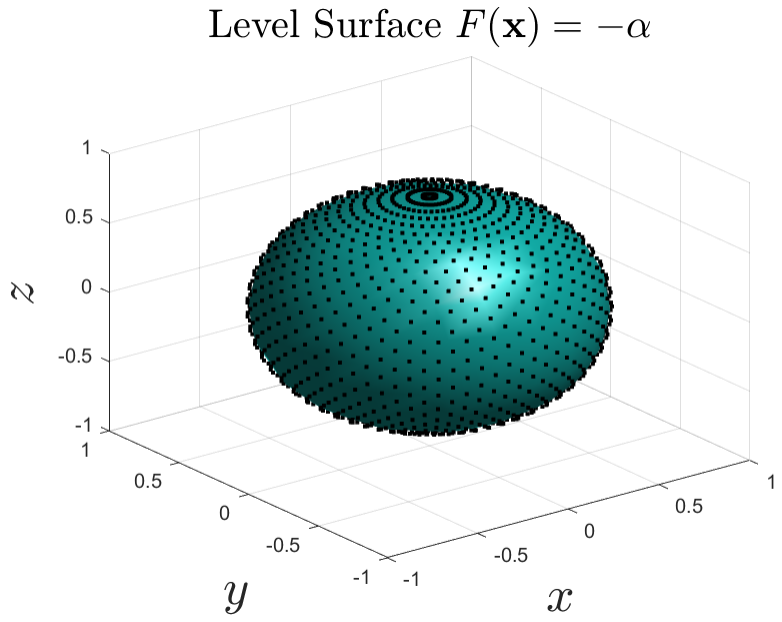


Figure 6: Level surface $F(x) = -\alpha$ of interpolating function utilizing IMQ RBFs. Black dots are of the x^- point cloud.

Exercise 3: RBF using pseudo-spectral Method

(a) Apply the RBF-PS method to solve the heat equation and Helmholtz equation (eigenvalue problem for the Laplacian) in 1D. (The relevant material is Chapter 42-43 in Fasshauer).

For the heat equation implementation, we seek to solve

$$u_t = u_{xx}$$

on the domain $x \in [0, 1]$ with initial and boundary conditions

$$u(x, 0) = e^{-80x^2}; \quad u(1, t) = u(0, t) = 0$$

Approximating the function u as a finite series of a radial basis function $\phi(x)$ yields

$$\begin{bmatrix} & \phi(\|x_i - x_j\|) & \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

and so

$$\begin{bmatrix} & \phi''(\|x_i - x_j\|) & \end{bmatrix} \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} u_1'' \\ \vdots \\ u_n'' \end{bmatrix}$$

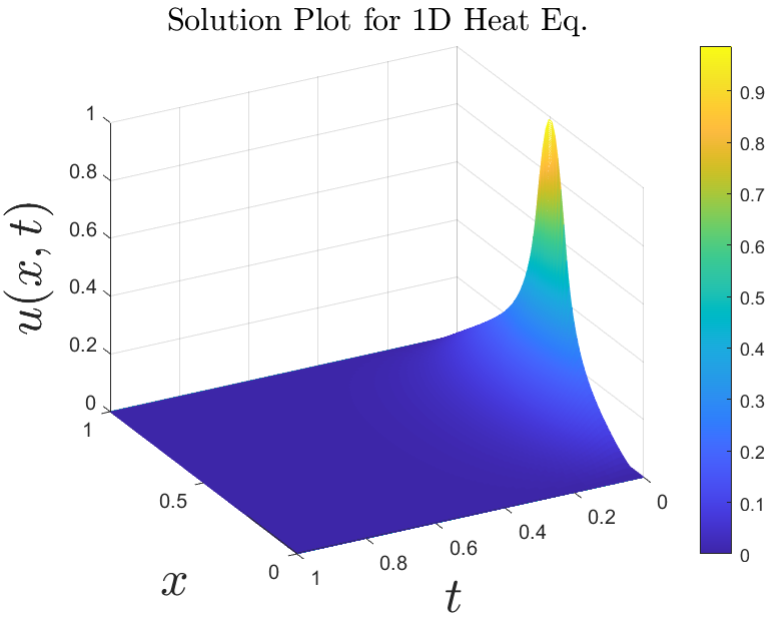
For this problem, the inverse multiquadric RBF was utilized:

$$\phi(x) = \frac{1}{\sqrt{1 + (\epsilon x)^2}}.$$

With Euler time stepping:

$$u_t(x_n, t_n) \approx \frac{u(x_n, t_n) - u(x_n, t_{n-1})}{dt}.$$

Implementing this in `MATLAB` (see attached `heateqRBF.m` script), we find the following solution plot for $t \in [0, 1]$:



Now, for the 1D Helmoltz eigenvalue problem, we wish to solve the eigenvalue problem

$$\frac{d^2u}{dx^2} = \lambda u$$

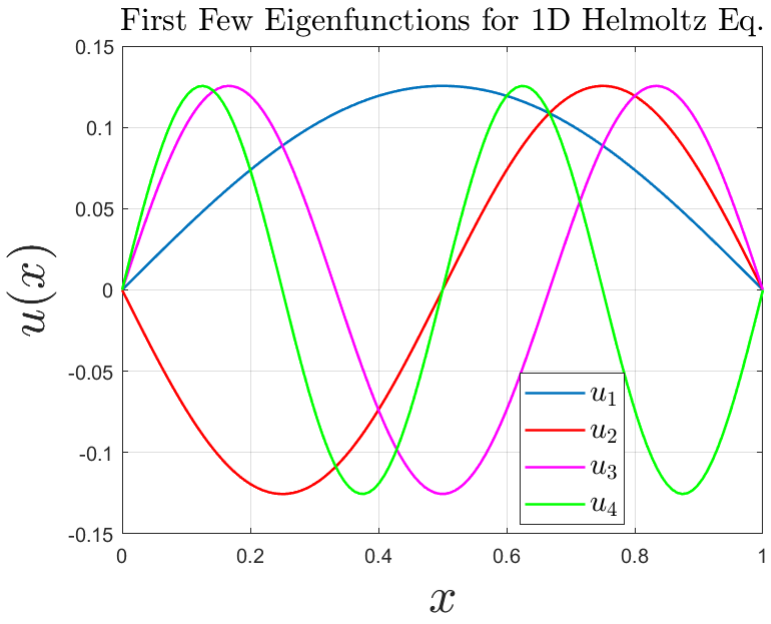
Solving analytically yields the eigenvalues

$$\lambda_n = -n^2\pi^2, \quad n \in \mathbb{N}$$

and eigenfunctions

$$u_n(x) = \sin(\sqrt{|\lambda_n|}x)$$

Using the differentiation matrix we found to solve the heat equation (and the built-in `MATLAB eig` command), we find the the following eigenvalues and eigenfunctions:

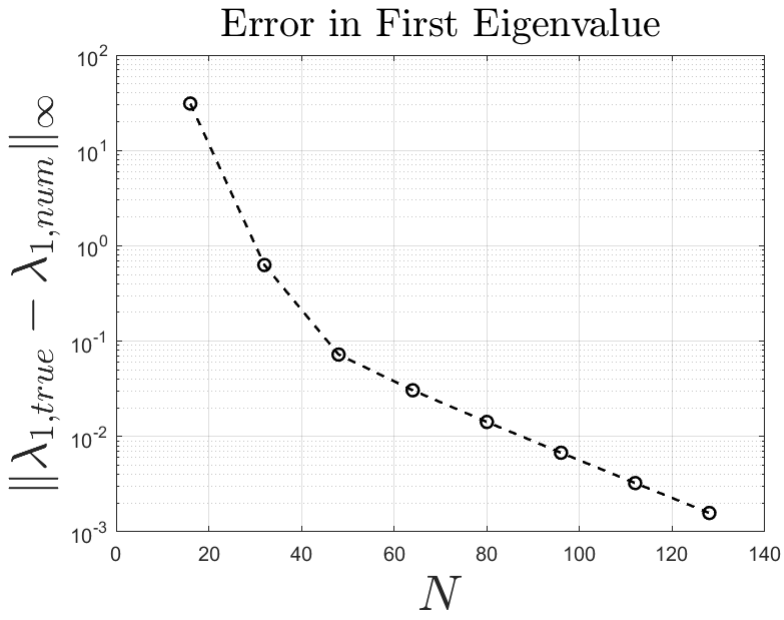



```
>> d(1:4)

ans =

   -9.8712
  -39.4834
  -88.8369
 -157.9311
```

And an associated error plot for the first eigenfunction as we vary the number of spacial points N:



(b) Describe the RBF-PS method in 2D (the implementation is optional).

For this problem, we will consider the 2D heat equation and the 2D wave equation. Recall that the 2D heat equation takes the form

$$u_t = k\nabla^2 u$$

with ∇^2 denoting the Laplacian operator. And the wave equation takes the form

$$u_{tt} = c^2\nabla^2 u$$

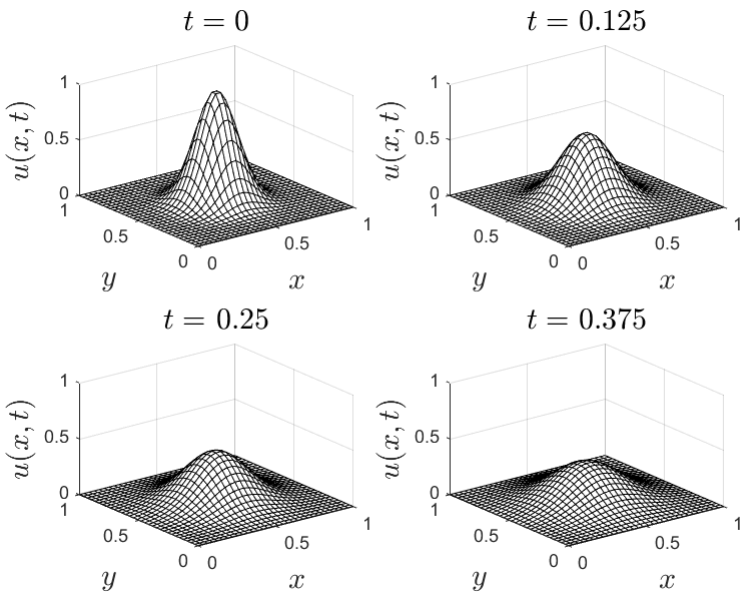
where c is the wave speed. For our implementation, we will take $k = 1/32$ and $c = 1$. For both equations, we will use Gaussian initial conditions with homogeneous Dirichlet boundary conditions. In particular, for the heat equation, we used the initial condition

$$u(x, y, 0) = e^{-40((x-1/2)^2+(y-1/2)^2)}$$

with $x \in [0, 1]$ and $y \in [0, 1]$.
To extend to 2D, our function $u(x, t)$ will be stored in an $N \times N$ matrix with the rows storing the x values and the columns storing the y values. To utilize matrix multiplication to approximate the derivative, we reshape the u matrix to be $N^2 \times 1$. Then given the second derivative RBF differentiation matrix, our Laplacian operator matrix will be computed by

$$L = D2 \otimes I + I \otimes D2$$

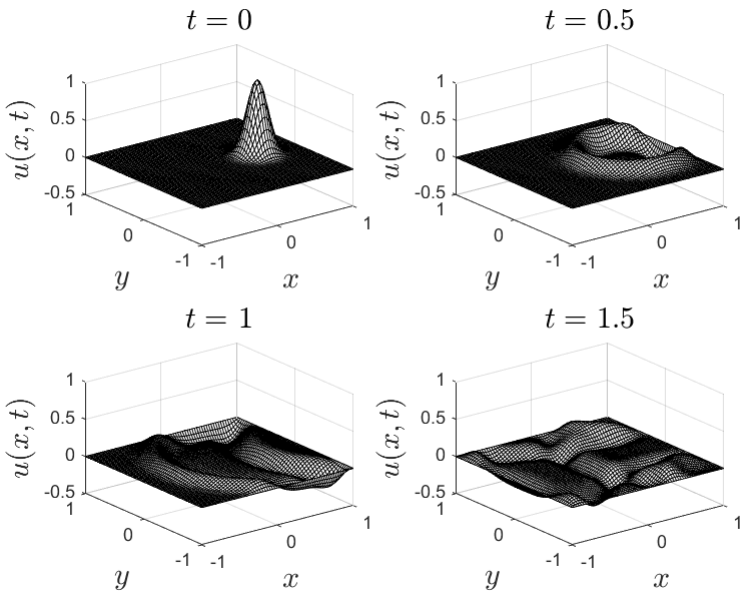
with I being an $N \times N$ identity matrix, and $\otimes$ denoting the Kronecker product. Implementing this in `MATLAB` (along with Euler time stepping), we find the following solution plots for heat equation for selected time values:



Now, for the wave equation, we use the initial condition

$$u(x, y, 0) = e^{-40((x-1/2)^2+y^2)}$$

with $x \in [-1, 1]$ and $y \in [-1, 1]$.
Implementing into `MATLAB` (along with the second order approximation $u_{tt} \approx \frac{u(x,y,t_{n+1})-2u(x,y,t_n)+u(x,y,t_{n-1})}{\Delta t^2}$), we find the following solution plots for selected time values:



For additional details, see the attached `twoDheatEqRBFTest.m` and `twoDwaveEqTestRBF.m` files.