Montestigliano Workshop | **Problem 1 RBF Interpolation on the Sphere** | **Level 1 Grady Wright**

In this problem you will implement your own code for interpolating data on the sphere using RBFs. The goal is for you to become familiar with various aspects of RBF interpolation, including the convergence rates one can expect for reconstructing target functions of varying smoothness, how the convergence rates change for kernels of different smoothness, what happens with different node sets, the effect of the shape parameter, and how to visualize the interpolants. The material for this problem comes from the first set of notes.

Note that the material in these notes discusses interpolation on the sphere using shifts of a zonal kernel $\psi : [-1, 1] \to \mathbb{R}$ expressed as $\psi(\mathbf{x}^T\mathbf{x}_j)$, where $\mathbf{x}, \mathbf{x}_j \in \mathbb{S}^2$, with $\mathbf{x}_j$ being the center of the kernel. However, RBF interpolants use shifts of a radial kernel $\phi : [0, \infty) \to \mathbb{R}$ expressed as $\phi(\|\mathbf{x} - \mathbf{x}_j\|)$, where $\mathbf{x}, \mathbf{x}_j \in \mathbb{R}^d$, $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^d$, and $\mathbf{x}_j$ is again the center. Of course, RBF interpolation is a special kind of zonal interpolant since for $\mathbf{x}, \mathbf{x}_j \in \mathbb{S}^2$,

$$\|\mathbf{x} - \mathbf{x}_j\| = \sqrt{1 - \mathbf{x}^T\mathbf{x}_j}$$

It is recommend that when you write your code for doing RBF interpolation on the sphere, you express everything in terms of ϕ and the standard 2-norm as shown on Slides 16 – 32 of the introductory lecture slides. This will be important when developing the code to compute surface derivatives, or if you want to use the code for interpolation on surfaces other than $\mathbb{S}^2$.

1.a) Write a MATLAB function for interpolating samples of some target function $f : \mathbb{S}^2 \to \mathbb{R}$ at a given set of nodes $X = \{\mathbf{x}\}_{j=1}^N \subset \mathbb{S}^2$ using the basic form of an RBF interpolant. Implement the code so that it accepts as input an array containing the nodes X and a vector containing the corresponding samples of f at X: $\{f_j\}_{j=1}^N$. The code should also take as input an array of M values where the interpolant is to be evaluated. To make the code even more general, also have it accept a function handle for the radial kernel that should be used for constructing the interpolant. Try to vectorize the code so that at most only one loop is used. Use Gaussian elimination (the `mldivide` or "backslash" function in MATLAB) to solve the linear system for the coefficient vector. In the case where the interpolation matrix is positive definite (i.e. ϕ is a positive definite kernel) you may consider instead using Cholesky factorization.

(i) Test your code for the inverse multiquadric (IMQ) radial kernel $\phi(r) = 1/\sqrt{1 + (\varepsilon r)^2}$, the target function

$$f(\mathbf{x}) = f(x, y, z) = \cos\left(2\left(x + \frac{1}{2}\right)^2 + 3\left(y + \frac{1}{2}\right)^2 + 5\left(z - \frac{1}{\sqrt{2}}\right)^2\right), \tag{1}$$

and the Maximal Determinant (MD) node set of size $N = 900$. You can obtain these nodes from the function `getMaxDetNodes` that is part of the `rbfsphere` MATLAB package.

For the evaluation point $\mathbf{x} = -(1, 1, 1)/\sqrt{3}$ and a shape parameter of $\varepsilon = 1.5$, your function should return the value $-0.412396840\ldots$. How close is this to the true value of the target function?

(ii) Repeat part (i) for the Gaussian kernel $\phi(r) = \exp(-(\varepsilon r)^2)$. For the same evaluation point and $\varepsilon = 2.5$ your function should return the value $-0.412396635\ldots$. How close is this to the true value of the target function?

(iii) Repeat part (i) for the compactly supported Wendland kernel

$$\phi(r) = (1-\varepsilon r)_+^8(32(\varepsilon r)^3 + 25(\varepsilon r)^2 + 8(\varepsilon r) + 1), \tag{2}$$

which is positive definite on $\mathbb{R}^3$ (and hence also on $\mathbb{S}^2$) and C^6. Here

$$(\xi)_+ = \begin{cases} \xi & \text{for } \xi \geq 0 \\ 0 & \text{for } \xi < 0. \end{cases}$$

This function can be implemented in MATLAB as `max(xi,0)`. The value $1/\varepsilon = \delta$ is called the support radius of the kernel. For interpolation on the unit sphere, as ε increases from $1/2$ (or δ decreases below 2) the resulting interpolation matrix will become more and more sparse. For $\varepsilon = 1$ with this test, your function should return the value $-0.412405378\ldots$. How close is this to the true value of the target function?

1.b) Now check how well the interpolants reconstruct the target function as the shape parameter varies. To get a better feeling for this, measure the error associated with each interpolant (i.e. difference between interpolant and target function) at several locations on the sphere and take the max-norm of the resulting error vector. For these locations, choose 1000 Fibonacci nodes using the function `getFibonacciNode` from the `rbfsphere` package. For each of the IMQ, Gaussian, and Wendland radial kernels, compute the *relative* max-norm errors for $\varepsilon = 0.2, 0.3, \ldots, 2.9, 3.0$. Make a plot of the errors vs. ε using a logarithmic scale for the dependent variable, i.e. use a `semilogy` plot. What do you observe?

For the IMQ and GA kernels, MATLAB should respond with some warnings regarding the poor conditioning of the linear systems associated with computing the interpolants for several small values of ε, you should similarly see some erratic behavior in the error plots for these value of ε. In a later problem, you will use the RBF-QR algorithm to bypass this observed ill-conditioning. The Wendland kernel will similarly display conditioning problems, but not for the range of ε considered here.

1.c) Repeat part 1.a) above but now implement a function for the so called general RBF interpolant, which is typically associated with conditionally positive definite (CPD) kernels of order k. This interpolant is discussed for problems on $\mathbb{R}^d$ in slides 26–32 of the introductory slides and for problems on $\mathbb{S}^2$ in slides 45–47. In the case of $\mathbb{S}^2$, which is what your function should handle, the interpolant takes the form:

$$I_X f(\mathbf{x}) = \sum_{j=1}^{N} c_j \phi(\|\mathbf{x} - \mathbf{x}_j\|) + \sum_{\ell=1}^{k^2} d_\ell p_\ell(\mathbf{x}), \tag{3}$$

where $\{p_\ell\}_{\ell=1}^{k^2}$ is a basis for the spherical harmonics of degree $k-1$. The interpolation conditions are $I_X f(\mathbf{x}_i) = f_i$, $i = 1, \ldots, N$ and $\sum_{j=1}^{N} c_j p_\ell(\mathbf{x}_j) = 0$, $\ell = 1, \ldots, k^2$. If you are not feeling up to writing a general code for any k, it is sufficient for what we do later to just consider $k = 1$ and $k = 2$, in these cases the elements of the basis can be simply chosen as

$$\begin{aligned} k = 1: &\quad p_1(\mathbf{x}) = 1 \\ k = 2: &\quad p_1(\mathbf{x}) = 1,\ p_2(\mathbf{x}) = x,\ p_3(\mathbf{x}) = y,\ p_4(\mathbf{x}) = z, \end{aligned}$$

which happen to be the same standard choice for problems on $\mathbb{R}^3$. If you do decide to write the more general case, then the function `sphHarm` in the `rbfsphere` package can be used to compute spherical harmonics of a given degree and order.

The general RBF interpolant is usually associated with conditionally positive definite radial kernels since it bypasses some ill-posedness issues associated with the basic RBF interpolant constructed

from these kernels. However, it can sometimes be useful to also use this form when working with positive definite kernels such as those from part 1.a) . The reason is that including the additional k^2 spherical harmonics in (3) guarantees that the interpolant will be exact for target functions that are any linear combination of spherical harmonics up to degree $k-1$. Typically in applications the benefit comes from using $k = 1$ or 2 so that it exactly reproduces constants and "linear" functions.

Your new MATLAB function for the general RBF interpolant should take the same input as your function from part 1.a), with the addition of the order parameter k.

(i) Test your code for the thin plate spline kernel $\phi(r) = r^2 \log r$ using $k = 2$ (which is sometimes called a surface spline of order 2 when restricted to problems on $\mathbb{S}^2$). This is a conditionally positive definite kernel of order 2 with two continuous derivatives. It does not depend on a shape parameter like the kernels used above. Use the same target function given in (1) above and the MD node set of size $N = 900$. For the evaluation point $\mathbf{x} = -(1,1,1)/\sqrt{3}$, your function should return the value $-0.4087676618\ldots$. How close is this to the true value of the target function?

Be careful in your code to evaluate ϕ correctly at $r = 0$ (it should be zero there). If you blindly plug in $\phi(0)$, MATLAB will most likely return a `nan`.

(ii) Repeat part (i) for the cubic kernel $\phi(r) = r^3$ using $k = 2$. This is also a conditionally positive definite kernel of order 2, but it has three continuous derivatives. For the evaluation point $\mathbf{x} = -(1,1,1)/\sqrt{3}$, your function should return the value $-0.4118687202\ldots$. How does this compare to the surface spline from (i)?

(iii) Repeat part (i) for the IMQ kernel with $k = 1$. For the evaluation point $\mathbf{x} = -(1,1,1)/\sqrt{3}$ and a shape parameter of $\varepsilon = 1.5$, your function should return the value $-0.412396840\ldots$. How close is this to the true value of the target function? You will not see much difference from the results from part (i) of problem 1.a) above.

1.d) In this problem you will check how the interpolants converge to the target function as the number of nodes in X increases. For this experiment use MD node sets of size $N = 10^2, 20^2, \ldots, 50^2$ and compute approximations to both the relative L_2 and L_∞ norms of the errors in the interpolant by evaluating them at $M = 2000$ Fibonacci nodes and computing the discrete relative ℓ_2 and ℓ_∞, i.e.

$$\text{Relative } \ell_2 \text{ error} = \frac{\sqrt{\sum_{i=1}^{M} |I_X f(\mathbf{x}_i) - f(\mathbf{x}_i)|^2}}{\sqrt{\sum_{i=1}^{M} |f(\mathbf{x}_i)|^2}}$$

$$\text{Relative } \ell_\infty \text{ error} = \frac{\max_{1 \le i \le M} |I_X f(\mathbf{x}_i) - f(\mathbf{x}_i)|}{\max_{1 \le i \le M} |f(\mathbf{x}_i)|}$$

Note that this is a crude approximation to the L_2 error on the sphere as it should involve a discrete approximation to the surface integral over the sphere using some quadrature formula. Later we will look at how to compute quadrature formulas on the sphere using RBFs. You will look at the convergence rates with respect to $\sqrt{N}$. Since the MD nodes are quasi-uniformly distributed on the sphere and the sphere is a two-dimensional surface, the spacing between the nodes goes down like $1/\sqrt{N}$.

(i) For this part you will look at the convergence rates for reconstructing the smooth target function f in (1). Compute the relative ℓ_2 and ℓ_∞ errors for the values of N listed above for the IMQ kernel with $\varepsilon = 1.75$, GA kernel with $\varepsilon = 4$, Wendland kernel (2) with $\varepsilon = 1$, and the thin plate spline kernel with $k = 2$. Produce plots of the various errors versus $\sqrt{N}$. For the IMQ and Gaussian you may want to use a `semiology` plot, while for the Wendland and thin plate spline kernels you may want to use a `loglog` plot. Note the differences in convergence rates between the infinitely smooth and finitely smooth kernels. Can you estimate how the convergence is

proceeding in each case? How is this related to the smoothness of the kernels? Which kernel performs the best and worst.

(ii) Repeat (i) above, but now for the non-smooth target function

$$f(\mathbf{x}) = f(x, y, z) = \begin{cases} z & \text{for } z \geq 0 \\ 0 & \text{for } z < 0 \end{cases} \tag{4}$$

which is only C^1. You may want to do all convergence plots using a `loglog` plot for this target function. Note the differences between the results from part (ii).

1.e) One of the primary purposes of constructing an interpolant to "scattered" data is to produce a plot of the data. Surface plots using the `surf` command in MATLAB require the data be sampled from some logically rectangular grid. In the case of the sphere, one can obtain such a grid using the command

```
[X,Y,Z] = sphere(m);
```

In this case `X`,`Y`, and `Z` will be matrices of size $(m+1)$-by-$(m+1)$ corresponding to $(m+1)^2$ gridded points on the sphere, where the grid consists of $m+1$ equally spaced points in longitude and $m+1$ equally spaced points in latitude. You can visualize the function (1) on the sphere using the commands:

```
f = @(x,y,z) cos(2*(x+1/2).^2+3*(y+1/2).^2+5*(z-1/sqrt(2)).^2);
[X,Y,Z] = sphere(200);
surf(X,Y,Z,f(X,Y,Z));
colorbar, shading flat, axis equal;
```

The last two lines just beautify the plot. It is sometimes useful to also make surface plots of functions in longitude and latitude. You can do that with the code below:

```
[L,T] = meshgrid(linspace(-pi,pi,201),linspace(-pi/2,pi/2,101));
[X,Y,Z] = sph2cart(L,T,1);
pcolor(L,T,f(X,Y,Z));
colorbar, shading flat, daspect([1 1 1]);
```

Another common technique for visualizing data on the surface of the sphere is to plot it on the Hammer map, which is an equal area map projection of the sphere developed by Ernst Hammer. The function `sph2hammer` in the `rbfsphere` package computes this mapping. The code below plots the function defined above on the Hammer map (assuming you have run the code above), it also adds in a plot of the coastlines of Earth so you can get a feel for what this type of projection does:

```
[XH,YH] = sph2hammer(L,T);
pcolor(XH,YH,f(X,Y,Z));
colorbar, shading flat, daspect([1 1 1]);
hold on;
plotCoastLines(2,'k-');
```

Note that since the function f is defined in Cartesian coordinates, it must be evaluated using these coordinates.

In practice, the target function f is unknown and only samples are given at some scattered locations. In this case you can interpolate the data using RBFs and then evaluate the interpolant at the gridded values in `X`,`Y`, and `Z`.

Produce plots of the target functions (1) and (4) by interpolating samples of these functions at $N = 900$ MD nodes. Try this for some of the different radial kernels you used in the examples above. Also make plots of the difference between the interpolants and the target functions. Where is the error the largest in the case of the non-smooth target (4)? How do the error plots change when you switch between using a infinitely smooth kernel (e.g. IMQ) and a finitely smooth kernel (e.g. thin plate spline)?

1.f) Repeat the previous experiment using some different node sets available from the `rbfsphere` package with the functions `getCubedSphNodes`, `getHealPixNodes`, `getMinEnergyNodes`, `getFibonacciNodes`, `getIcosNodes`, and `getHammersleyNodes`. How do the plots of the errors change when using these node sets?

> Be careful to read the help files for the node functions listed above. The input argument is not always equal to the total number of nodes you request, as some of the node sets have special structure that only allows certain values of N.

1.g) (Optional) *Why are some kernels called conditionally positive definite?* Recall that a radial kernel is positive definite on $\mathbb{S}^2$ if for any distinct set of nodes $\{\mathbf{x}_j\}_{j=1}^N \subset \mathbb{S}^2$, the N-by-N matrix A with entries $A_{i,j} = \phi(\|\mathbf{x}_i - \mathbf{x}_j\|)$ is positive definite. A conditionally positive definite kernel is one that is not positive definite on all of $\mathbb{R}^N$, but on a special subspace of $\mathbb{R}^N$ related to properties of the kernel and, in the case of the sphere, spherical harmonics. If a radial kernel ϕ is conditionally positive definite of order k and $\{p_\ell\}_{\ell=1}^{k^2}$ is a basis for the spherical harmonics of degree $k-1$, then the subspace that ϕ is positive definite on is

$$\mathcal{P}_\perp = \left\{ \mathbf{b} \in \mathbb{R}^N \;\middle|\; \mathbf{b} \neq 0 \text{ and } \sum_{j=1}^{N} b_j p_\ell(\mathbf{x}_j) = 0, \text{ for } \ell = 1, \ldots, k^2 \right\}. \tag{5}$$

This subspace is denoted as $\mathcal{P}_\perp$ because it is the orthogonal complement of the space spanned by all linear combinations of the vectors $\mathbf{p}_\ell$, $\ell = 1, \ldots, k^2$, where $(\mathbf{p}_\ell)_j = p_\ell(\mathbf{x}_j)$ for $j = 1, \ldots, N$, i.e.

$$\mathcal{P} = \text{span}\left\{ \begin{bmatrix} p_1(\mathbf{x}_1) \\ \vdots \\ p_1(\mathbf{x}_N) \end{bmatrix}, \begin{bmatrix} p_2(\mathbf{x}_1) \\ \vdots \\ p_2(\mathbf{x}_N) \end{bmatrix}, \ldots, \begin{bmatrix} p_{k^2}(\mathbf{x}_1) \\ \vdots \\ p_{k^2}(\mathbf{x}_N) \end{bmatrix} \right\}. \tag{6}$$

Verify this numerically in the case of the thin plate spline $\phi(r) = r^2 \log r$ which is conditionally positive definite of order 2.

Choose a set of nodes on the surface of the sphere using, for example, one of the functions from the `rbfsphere` package (e.g. `getMaxDetNodes`). Construct the N-by-N matrix A with entries $A_{i,j} = \phi(\|\mathbf{x}_i - \mathbf{x}_j\|)$ and compute its eigenvalues using the `eig` function in MATLAB , e.g.

```
ev = eig(A);
```

Verify that there are 3 negative eigenvalues and the rest are positive (all the eigenvalues are real since A is symmetric). Hence, the matrix is not positive definite.

Next, construct the following N-by-4 matrix P

$$P = \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & y_N & z_N \end{bmatrix}$$

For the thin plate spline, the column space of P is equal to $\mathcal{P}$ in (6) since $1, x, y, z$ are a basis for all spherical harmonics of degree 1.

Now, construct the N-by-N matrix $Q_\perp$ for projecting any vector $\mathbf{b} \in \mathbb{R}^N$ onto the space $\mathcal{P}_\perp$ in (5). This can be done with the SVD using the following MATLAB commands:

```
[U,S,V] = svd(P);
Qperp = U(:,5:end)*U(:,5:end).';
```

According to the theory, the matrix $Q_\perp A Q_\perp$ should be positive semi-definite. Verify that this is the case by computing the eigenvalues of this matrix using the `A` and `Qperp` computed above.

Note $Q_\perp AQ_\perp$ is symmetric and should have all real eigenvalues, but this property may not hold after numerically forming the product, so you may need to check only the real part of the eigenvalues of this matrix. Also, all eigenvalues should be non-negative up to rounding errors.

The zero eigenvalues of $Q_\perp AQ_\perp$ correspond to eigenvectors that are in the column space of P. For any vector $\mathbf{b}$ not in this space the quadratic form $\mathbf{b}^T(Q_\perp AQ_\perp)\mathbf{b}$ will be positive.