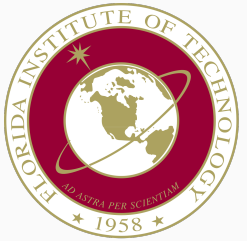


# Radial Basis Function and Shephard Interpolation

Stephen D. Butalla & Valerie Kobzarenko

September 21, 2019

CSE-5400
Florida Institute of Technology

# Introduction: Radial Basis Function Interpolation

- Method for $N$-dimensional interpolation
- Resource intensive: $\mathcal{O}\left(n^3\right)$ to perform LU decomposition to find multiplicative weights for the RBF, and $\mathcal{O}(n)$ operations to find each interpolated value
- If the situation allows, choose another method:
  1. If the dimension of the space is $n > 2$, and the data are relatively dense, consider using triangulation (a finite element method, discussed in § 21.6 of [1]), or
  2. If accuracy is not a concern, then transfer data to Cartesian coordinates and use Laplace interpolation

# Concept

- Assume every grid point $j$ influences it's neighboring coordinates $\Rightarrow$ approximate the interpolating function that depends only on radial distance: $\phi(r) = \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)$
- The interpolating function $\Gamma(\mathbf{x})$ can be expressed as a weighted sum of these radial basis functions (RBFs):

$$\Gamma(\mathbf{x}) = \sum_{i=0}^{N-1} w_i \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)$$

  where $w_i$'s are unknown weights. Note that the argument of $\Gamma(\mathbf{x})$ is in general an $n$ dimensional vector!
- Weights are found by requiring the interpolation to be exact at each grid point (i.e., having a value), and solving a set of $N$ linear equations:

$$\Gamma_j = \sum_{i=0}^{N-1} w_i \phi\big(|\mathbf{x}_j - \mathbf{x}_i|\big) \qquad \Rightarrow \qquad \boldsymbol{\Gamma} = \boldsymbol{\Phi}\mathbf{w}$$

# Concept

- Normalized radial basis function (NRBF) is a similar method (scaling $\Gamma(\mathbf{x})$ to unity)
- The interpolating function $\Gamma(\mathbf{x})$ can be expressed as a weighted sum of these radial basis functions (RBFs):

$$\Gamma(\mathbf{x}) = \frac{\sum_{i=0}^{N-1} w_i \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)}{\sum_{i=0}^{N-1} \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)}$$

- Neither RBF or NRBF is better than the other

# Radial Basis Functions: Multiquadratic

- What does an RBF look like in practice?
- Multiquadratic

$$\phi(r) = \left(r^2 + r_0^2\right)^{1/2}$$

where $r$ is the Euclidean "radial" distance in $n$ dimensions

$$r = |\mathbf{x} - \mathbf{x}_i| = \sqrt{\left(x_1 - x_1^{(i)}\right)^2 + \left(x_2 - x_2^{(i)}\right)^2 + \cdots + \left(x_n - x_n^{(i)}\right)^2}$$

and $r_0^2$ is a free parameter (scaling factor)

- Most commonly used RBF
- Choice of $r_0$ doesn't influence this RBF as much as others

# Radial Basis Functions: Choosing $r_0$

- $r_0$ should be:
  - Larger than average separation of coordinates
  - Smaller than the "outer" scale of the object you are interpolating [1]
- Choice of $r_0$ is important: interpolation accuracy can be off by orders(!) of magnitude for poorly chosen $r_0$
- To find an appropriate $r_0$, one can omit a data point $j$ and measure the interpolation error at $j$

# Radial Basis Functions: Other Functions

- Inverse multiquadratic

$$\phi(r) = \left(r^2 + r_0^2\right)^{-1/2}$$

  Equal to or better results than multiquadratic; useful if you need extrapolated function $f(x) \rightarrow 0$ as $x \rightarrow \infty$

- Thin-plate spline

$$\phi(r) = r^2 \log\left(\frac{r}{r_0}\right)$$

- Gaussian

$$\phi(r) = \exp\left(\frac{r^2}{2r_0^2}\right)$$

  Very sensitive to $r_0$, but with smooth functions interpolation can be very accurate

# RBF Algorithm

- Input coordinate data as an $n$ dimensional matrix `pts` and function values as `vals` and fill the $\phi$ matrix using

$$\phi_{ij} = \phi(|\mathbf{x}_i - \mathbf{x}_j|)$$

- Use LU decomposition to solve for the weights $w_i$

```
for (i=0;i<n;i++) {
  sum = 0.;
  //fills phi matrix and rhs vector
  for (j=0;j<n;j++) {
    sum += (rbf[i][j]
    = fn.rbf(rad(&pts[i][0],&pts[j][0])));
  }//rad computes Euclidean distance
   //fn.rbf() calls chosen RBF function
  if (norm) rhs[i] = sum*vals[i];
  else rhs[i] = vals[i];
}
LUdcmp lu(rbf); //LU decomp constructor
lu.solve(rhs,w); //Solve system with LU; return w
```

# RBF Algorithm

- rad function:

```
Doub rad(const Doub *p1, const Doub *p2) {
  Doub sum = 0.;
  for (Int i=0;i<dim;i++) sum += SQR(p1[i]-p2[i]);
  return sqrt(sum);
}
```

$$r = \sqrt{\left(x_1 - x_1^{(i)}\right)^2 + \left(x_2 - x_2^{(i)}\right)^2 + \cdots + \left(x_n - x_n^{(i)}\right)^2}$$

- Thing-plate spline example constructor $\phi(r)$

```
struct RBF_thinplate : RBF_fn {
 Doub r0;
 RBF_thinplate(Doub scale=1.) : r0(scale) {}
 Doub rbf(Doub r) { return r <= 0. ? 0. : SQR(r)*log(r/r0); }
};
```

$$\phi(r) = r^2 \log\left(\frac{r}{r_0}\right)$$

# RBF Algorithm

- Finally, call `interp` to find interpolated value at `pt[]`, i.e., coordinate $(x_1, x_2, \cdots, x_n)$ using the original expression

$$\Gamma(\mathbf{x}_{\text{int}}) = \sum_{i=0}^{N-1} w_i \phi\big(|\mathbf{x}_{\text{int}} - \mathbf{x}_i|\big)$$

```
Doub interp(VecDoub_I &pt) {
 Doub fval, sum=0., sumw=0.;
 if (pt.size() != dim) throw("RBF_interp bad pt size");
 for (Int i=0;i<n;i++) {
   fval = fn.rbf(rad(&pt[0],&pts[i][0]));
   sumw += w[i]*fval;
   sum += fval;
 }
return norm ? sumw/sum : sumw;
//if norm is true, return sumw/sum, else sumw
}
```

# RBF Example

- Given a set of data points:

$$\mathbf{x} = \{0.3086, 1.5812, 0.387417\}$$
$$\mathbf{y} = \{0.84649, 0.492773, 0.338693\}$$
$$\mathbf{z} = \{1.50064, 3.94129, 1.14157\}$$

  we first find the $\mathbf{\Phi}$ matrix using our choice of RBF and scale factor
- The input coordinate matrix is then

$$\mathbf{\Phi} = \begin{pmatrix} 0.3086 & 0.84649 \\ 1.5812 & 0.492773 \\ 0.387417 & 0.338693 \end{pmatrix}$$

  and the $\mathbf{\Gamma}$ vector is

$$\mathbf{\Gamma}^T = (1.50064 \;\; 3.94129 \;\; 1.14157)$$

- Let

$$\phi(r) = \left(r^2 + r_0^2\right)^{1/2}$$

  where $r_0 = 2$

# RBF Example

- First, find the Euclidean distances between points:

$$r_{ij} = |\mathbf{x}_i - \mathbf{x}_j| = \sqrt{\left(x_i^{(0)} - x_j^{(0)}\right)^2 + \cdots + \left(x_i^{(n)} - x_j^{(n)}\right)^2}$$

  and calculate the matrix element $\phi_{ij}(r_{ij})$
- Example:

$$\begin{aligned} r_{ij} &= |\mathbf{x}_1 - \mathbf{x}_0| \\ &= \sqrt{\left(x_1^{(0)} - x_0^{(0)}\right)^2 + \left(x_1^{(1)} - x_0^{(1)}\right)^2} \\ &= \sqrt{(1.5812 - 0.3086)^2 + (0.492773 - 0.84649)^2} \\ &= 1.320843 \end{aligned}$$

  Now,

$$\begin{aligned} \phi_{ij}(r_{ij}) &= \left(r_{ij}^2 + r_0^2\right)^{1/2} \\ &= \left((1.320843)^2 + (2)^2\right)^{1/2} \\ &= 5.744626 \end{aligned}$$

# RBF Example

- Construct the $\mathbf{\Phi}$ matrix:

$$\mathbf{\Phi} = \begin{pmatrix} 0 & 5.74 & 0.51 \\ 5.74 & 0 & 1.20 \\ 0.51 & 1.20 & 0 \end{pmatrix}$$

and use LU decomposition to find the weights:

$$\mathbf{\Gamma} = \mathbf{\Phi w}$$

$$\begin{pmatrix} 1.50064 \\ 3.94129 \\ 1.14157 \end{pmatrix} = \begin{pmatrix} 0 & 5.74 & 0.51 \\ 5.74 & 0 & 1.20 \\ 0.51 & 1.20 & 0 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

$$\mathbf{w}^T = (1.15505 \;\; 0.460412 \;\; -2.24059)$$

# RBF Example

- Finally, we can interpolate a point at $\mathbf{x}_{\text{int}}$ by using the original expression:

$$\Gamma(\mathbf{x}) = \sum_{i=0}^{N-1} w_i \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)$$

So,

$$\Gamma(\mathbf{x}_{\text{int}}) = \sum_{i=0}^{N-1} w_i \phi\big(|\mathbf{x}_{\text{int}} - \mathbf{x}_i|\big)$$

Using the point $\mathbf{x}_{\text{int}} = (0, 1)$ and evaluating the sum above, we find

$$\Gamma(\mathbf{x}_{\text{int}}) = 1.24557$$

# Shephard Interpolation

- Shephard Interpolation is closely related to RBF
- Uses NRBF with the choice of the RBF as a power law:

$$\Gamma(\mathbf{x}) = \frac{\sum_{i=0}^{N-1} w_i \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)}{\sum_{i=0}^{N-1} \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)}$$

  with

$$\phi(r) = r^{-p}$$

  where $1 \leq p \leq 3$ is typically used
- Because the function is a power law, $\phi \to 0$ as $r \to \infty$
- In this case, the weights are simply the function values: $w_i = \Gamma_i$

$$\Gamma(\mathbf{x}) = \frac{\sum_{i=0}^{N-1} \Gamma_i \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)}{\sum_{i=0}^{N-1} \phi\big(|\mathbf{x} - \mathbf{x}_i|\big)}$$

- Note that no set of linear equations needs to be solved $\Rightarrow$ the order of complexity is reduced: $\mathcal{O}(n^3) \to \mathcal{O}(n)$

# Shephard Interpolation

- This method is essentially interpolation "by a nearness-weigthed average" [1]
- Because of it's speed, it is useful for quick applications
- However, it lacks accuracy compared to other RBFs

# References

[1] W. Press, S. Teukolsky, W. Vetterling, & B. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 2007, Cambridge University Press.