

GM with RBFs for IE and PDE-Numerical Experiment

Abstract

This paper investigates the power of Radial Basis Functions(RBFs) to solving numerically Integral Equations(IE) and Partial Differential Equations(PDE) in the sense of Galerkin Method(GM). Galerkin method with radial basis functions is a global one just like spectral method, but it has the flexibility of Finite Element Method(FEM).

1 Introduction

As a powerful approximation method, radial basis functions have played important roles in Dual Reciprocity boundary element Method(DRM)[1] and Artificial Intelligence(AI)[2]. Since radial basis functions have desirable approximation properties in spaces of continuous functions and Sobolev spaces[3, 4, 5], Can we expand solutions of integral equations and boundary value problems in terms of radial basis functions? This thought leads us to produce a kind of global Galerkin method which uses radial basis functions as trial and test functions.

The commonly used radial basis functions $\phi(r)$ are

e^{-r^2}	Gaussian
r^3	cubic
$(1+r^2)^{\frac{1}{2}}$	multiquadric
$(1+r^2)^{-\frac{1}{2}}$	inverse multiquadric
r	linear and
$r^2 \ln(r)$	thin plate spline

If Ω is an open bounded subset in R^d , x_0, x_1, \dots, x_{n-1} are pairwise distinct points in $\bar{\Omega}$, $u(x)$ is a function from $\bar{\Omega}$ to R , we can expand $u(x)$ in terms of radial basis functions

$$u(x) \sim u_a(x) = \sum_{j=0}^{n-1} \lambda_j \phi\left(\frac{r_j}{\beta_j}\right) \quad (1)$$

where $r_j = \|x - x_j\|$ ($j = 0, 1, \dots, n-1$) are Euclidean norms, β_j ($j = 0, 1, \dots, n-1$) are constants. Sometimes, for the sake of stability, we can add a polynomial of k order to the expansion

$$u(x) \sim u_a(x) = \sum_{j=0}^{n-1} \lambda_j \phi\left(\frac{r_j}{\beta_j}\right) + p_k \quad (2)$$

2 For Integral Equation

If the second Fredholm equation

$$u(x) = f(x) + (Ku)(x) =: f(x) + \int_a^b k(x, y)u(y)dy \quad (3)$$

has unique solution, we use (2) to approximate $u(x)$ (if linear polynomial is employed in (2))

$$\begin{aligned} u_a(x) &= \sum_{j=0}^{n-1} \lambda_j \phi\left(\frac{r_j}{\beta_j}\right) + p_1 \\ &= \sum_{j=0}^{n-1} \lambda_j \phi\left(\frac{r_j}{\beta_j}\right) + \lambda_n + \lambda_{n+1}x \\ &= \sum_{j=0}^{n+1} \lambda_j \phi_j(x) \end{aligned} \quad (4)$$

here

$$\phi_j(x) = \begin{cases} \phi\left(\frac{r_j}{\beta_j}\right) & \text{if } 0 \leq j \leq n-1 \\ 1 & \text{if } j = n \\ x & \text{if } j = n+1 \end{cases} \quad (5)$$

By Galerkin method with RBFs being basis functions, we get linear equations

$$\sum_{j=0}^{n+1} \lambda_j (\phi_j, \phi_i) = (f, \phi_i) + \sum_{j=0}^{n+1} \lambda_j (K\phi_j, \phi_i) \quad i = 0, 1, \dots, n+1 \quad (6)$$

where (\cdot, \cdot) denotes inner product in L_2 , by means of matrix notations, (6) becomes

$$A\lambda = F + B\lambda \quad (7)$$

with

$$\begin{aligned} a_{ij} &= (\phi_j, \phi_i) & i, j = 0, 1, \dots, n+1 \\ b_{ij} &= (K\phi_j, \phi_i) & i, j = 0, 1, \dots, n+1 \\ f_i &= (f, \phi_i) & i = 0, 1, \dots, n+1 \end{aligned} \quad (8)$$

An example, with $[a, b] = [-1, 1]$, $k(x, y) = \frac{1}{1+x^2+y^2}$, $f(x) = x^2 - 2 + 2\sqrt{1+x^2} \arctan \frac{1}{\sqrt{1+x^2}}$, $\phi(r) = e^{-r^2}$, $\beta_j = 1$, is showed on Table 1.

3 For Elliptic Boundary Value Problem

Now, we consider Poisson equation

$$\begin{cases} -\Delta u(x) = f(x) & \text{in } \Omega \subset R^2 \\ u(x) = g(x) & \text{on } \Gamma = \partial\Omega \end{cases} \quad (9)$$

node	coordinate	approximation	exact-solution	relative-error(%)
0	-1.000000	0.999997	1.000000	0.000265
1	-0.894737	0.800551	0.800554	0.000333
2	-0.789474	0.623266	0.623269	0.000429
3	-0.684211	0.468141	0.468144	0.000574
4	-0.578947	0.335177	0.335180	0.000803
5	-0.473684	0.224374	0.224377	0.001203
6	-0.368421	0.135731	0.135734	0.001993
7	-0.263158	0.069249	0.069252	0.003912
8	-0.157895	0.024928	0.024931	0.010878
9	-0.052632	0.002767	0.002770	0.097947
10	0.052632	0.002767	0.002770	0.097947
11	0.157895	0.024928	0.024931	0.010878
12	0.263158	0.069249	0.069252	0.003912
13	0.368421	0.135731	0.135734	0.001993
14	0.473684	0.224374	0.224377	0.001203
15	0.578947	0.335177	0.335180	0.000803
16	0.684211	0.468141	0.468144	0.000574
17	0.789474	0.623266	0.623269	0.000429
18	0.894737	0.800551	0.800554	0.000333
19	1.000000	0.999997	1.000000	0.000265

Table 1: An Example of Integral Equation

Let $u_a(x)$ approach $u(x)$, and $u_a(x)$ has the form

$$\begin{aligned}
u_a(x) &= \sum_{j=0}^{n-1} \lambda_j \phi\left(\frac{r_j}{\beta_j}\right) + p_1 \\
&= \sum_{j=0}^{n-1} \lambda_j \phi\left(\frac{r_j}{\beta_j}\right) + \lambda_n + \lambda_{n+1}x + \lambda_{n+2} \\
&= \sum_{j=0}^{n+2} \lambda_j \phi_j(x)
\end{aligned} \tag{10}$$

where

$$\phi_j(x) = \begin{cases} \phi\left(\frac{r_j}{\beta_j}\right) & \text{if } 0 \leq j \leq n-1 \\ 1 & \text{if } j = n \\ x & \text{if } j = n+1 \\ y & \text{if } j = n+2 \end{cases} \tag{11}$$

then, $\{\lambda_j\}$ must satisfy the following linear system in the sense of Galerkin method

$$\begin{cases} \sum_{j=0}^{n+2} \lambda_j (\nabla \phi_j, \nabla \phi_i) &= (f, \phi_i) + (u_n, \phi_i)_\Gamma \\ \sum_{j=0}^{n+2} \lambda_j (x_i) &= g(x_i) \quad \text{if } x_i \in \Gamma \end{cases} \tag{12}$$

where $u_n(x)$ is the out normal derivative of $u(x)$, $(u_n, \phi_i)_\Gamma$ means $\int_\Gamma u_n \phi_i d\Gamma$.

For simplicity, we adopt constant elements to compute $(u_n, \phi_i)_\Gamma$. Suppose that there are m nodes on Γ , they are x_0, x_1, \dots, x_{m-1} sequentially, then

$$(u_n, \phi_i)_\Gamma \doteq \sum_{j=0}^{m-1} \mu_j (1, \phi_i)_{\Gamma_j} \quad (13)$$

where $\Gamma \doteq \sum_{j=0}^{m-1} \Gamma_j$, $\Gamma_j = x_j x_{j+1}$ with $x_m = x_0$, $\mu_j = u_n(\frac{x_j+x_{j+1}}{2})$ also with $x_m = x_0$. Simple calculation finds

$$\begin{bmatrix} A & -C \\ B & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \end{bmatrix} = \begin{bmatrix} F \\ G \end{bmatrix} \quad (14)$$

with

$$\begin{aligned} a_{ij} &= (\nabla \phi_j, \nabla \phi_i) & i, j = 0, 1, \dots, n+2 \\ b_{ij} &= \phi_j(x_i) & i = 0, 1, \dots, m-1, \quad j = 0, 1, \dots, n+2 \\ c_{ij} &= (1, \phi_i)_{\Gamma_j} & i = 0, 1, \dots, n+2, \quad j = 0, 1, \dots, m-1 \\ f_i &= (f, \phi_i) & i = 0, 1, \dots, n+2 \\ g_i &= g(x_i) & i = 0, 1, \dots, m-1 \end{aligned} \quad (15)$$

An example, with $\Omega = (-1, 1) \times (-1, 1)$, $g(x) = 1 + (x^1)^2 + (x^2)^2$, $f(x) = -4$, $\phi(r) = e^{-r^2}$, $\beta_j = 1$, is showed on Table 2.

4 For Time-Dependent Problem

Naturally, we can extend the above procedure to time-dependent problems, without loss of generality, two-dimensional parabolic equation is considered

$$\left\{ \begin{array}{l} \frac{\partial u(x,t)}{\partial t} - \Delta u(x,t) = f(x,t) \quad \text{in } \Omega \times (0, T) \\ u(x,t) = g(x,t) \quad \text{on } \Gamma = \partial\Omega \\ u(x,0) = h(x) \end{array} \right\} \quad (16)$$

If (1) is employed to approach $u(x, t)$ at the time t

$$u(x, t) \doteq \sum_{j=0}^{n-1} \lambda_j(t) \phi_j(x) \quad (17)$$

we, similarly to Section 3, get the following ordinary differential equations

$$\left\{ \begin{array}{l} D \frac{d\lambda}{dt} + A\lambda = F + C\mu \\ B\lambda = G \\ \lambda(0) = \lambda^h \end{array} \right\} \quad (18)$$

where A, B, C, F, G are similar to Section 3, $d_{ij} = (\phi_j, \phi_i)$, and the components of λ^h satisfy

$$h(x) \doteq \sum_{j=0}^{n-1} \lambda_j^h \phi_j(x) \quad (19)$$

If backward Euler method is employed, the full discrete scheme is

$$\begin{bmatrix} \Delta t A + D & -C \\ B & 0 \end{bmatrix} \begin{bmatrix} \lambda^{k+1} \\ \mu^{k+1} \end{bmatrix} = \begin{bmatrix} \Delta t F^{k+1} + D\lambda^k \\ G^{k+1} \end{bmatrix} \quad (20)$$

with initial condition

$$\lambda^0 = \lambda^h \quad (21)$$

An example, with $\Omega = (-1, 1) \times (-1, 1)$, $f(x, t) = -(4 + (x^1)^2 + (x^2)^2)e^{-t}$, $g(x, t) = 1 + ((x^1)^2 + (x^2)^2)e^{-t}$, $h(x) = 1 + (x^1)^2 + (x^2)^2$, $\phi(r) = e^{-r^2}$, $\beta_j = 1$, is showed on Table 3 and Table 4.

5 Conclusion

A numerical method for solving mathematical and physical problems has been investigated, some examples, which demonstrate the efficiency of the method, are given. The method is a kind of Galerkin method, it is very flexible for domain-shape and easy to implement. what is worthwhile to point out is that the choice of β_j is very important to the present method, too small values produce isolated peaks and too large values make the resulting linear equations ill-conditioned (the case is like [6]), but the criterion by which the β_j can be choosed properly has not been available yet, and that adding the polynomial p_k to the expansion obviously helps the stability of approximation procedure.

References

- [1] Partridge, P.W., Brebbia, C.A. & Wrobel, L.C. The Dual Reciprocity Boundary Element Method. (1992)
- [2] Girosi, Federico Some extensions of radial basis functions and their applications in artificial intelligence. Comput. Math. Appl. 24(1992), no. 12, p61-80
- [3] Powell, M.J.D. The theory of radial basis function approximation in 1990. Advances in numerical analysis, Vol. II (1992)
- [4] Dyn, N., & Ron, A. Radial basis function approximation: from gridded centres to scattered centres. Proc. London Math. Soc. (3) 71(1995) p76-108
- [5] Schaback, R. Approximation by radial basis functions with finitely many centers. Constr. Approx. 12(1996), no. 3, p331-340.
- [6] Yamada, T. & Wrobel, L.C. Properties of Gaussian radial basis functions in the dual reciproctiy boundary element method. Zangew Math. Phys. 44(1993), p1054-1067

node	x	y	appro	exact-	relative-error(%)
0	-1.000000	-1.000000	3.000000	3.000000	0.000000
1	-0.714286	-1.000000	2.510204	2.510204	0.000000
2	-0.428571	-1.000000	2.183673	2.183673	0.000000
3	-0.142857	-1.000000	2.020408	2.020408	0.000000
4	0.142857	-1.000000	2.020408	2.020408	0.000000
5	0.428571	-1.000000	2.183673	2.183673	0.000000
6	0.714286	-1.000000	2.510204	2.510204	0.000000
7	1.000000	-1.000000	3.000000	3.000000	0.000000
8	-1.000000	-0.714286	2.510204	2.510204	0.000000
9	-0.714286	-0.714286	2.020380	2.020408	0.001375
10	-0.428571	-0.714286	1.693958	1.693878	0.004771
11	-0.142857	-0.714286	1.530687	1.530612	0.004899
12	0.142857	-0.714286	1.530684	1.530612	0.004709
13	0.428571	-0.714286	1.693960	1.693878	0.004880
14	0.714286	-0.714286	2.020380	2.020408	0.001373
15	1.000000	-0.714286	2.510204	2.510204	0.000000
16	-1.000000	-0.428571	2.183673	2.183673	0.000000
17	-0.714286	-0.428571	1.693960	1.693878	0.004884
18	-0.428571	-0.428571	1.367290	1.367347	0.004133
19	-0.142857	-0.428571	1.204040	1.204082	0.003444
20	0.142857	-0.428571	1.204042	1.204082	0.003252
21	0.428571	-0.428571	1.367291	1.367347	0.004127
22	0.714286	-0.428571	1.693959	1.693878	0.004785
23	1.000000	-0.428571	2.183673	2.183673	0.000000
24	-1.000000	-0.142857	2.020408	2.020408	0.000000
25	-0.714286	-0.142857	1.530684	1.530612	0.004699
26	-0.428571	-0.142857	1.204043	1.204082	0.003209
27	-0.142857	-0.142857	1.040852	1.040816	0.003400
28	0.142857	-0.142857	1.040852	1.040816	0.003419
29	0.428571	-0.142857	1.204040	1.204082	0.003440
30	0.714286	-0.142857	1.530687	1.530612	0.004886
31	1.000000	-0.142857	2.020408	2.020408	0.000000

Table 2: An Example of Boundary Value Problem

node	x	y	appro	exact	relative-error(%)
0	-1.000000	-1.000000	1.898658	1.898658	0.000000
1	-0.714286	-1.000000	1.678578	1.678578	0.000000
2	-0.428571	-1.000000	1.531859	1.531859	0.000000
3	-0.142857	-1.000000	1.458499	1.458499	0.000000
4	0.142857	-1.000000	1.458499	1.458499	0.000000
5	0.428571	-1.000000	1.531859	1.531859	0.000000
6	0.714286	-1.000000	1.678578	1.678578	0.000000
7	1.000000	-1.000000	1.898658	1.898658	0.000000
8	-1.000000	-0.714286	1.678578	1.678578	0.000000
9	-0.714286	-0.714286	1.458892	1.458499	0.019472
10	-0.428571	-0.714286	1.313462	1.311779	0.099368
11	-0.142857	-0.714286	1.238680	1.238419	0.017018
12	0.142857	-0.714286	1.240022	1.238419	0.104679
13	0.428571	-0.714286	1.312863	1.311779	0.063995
14	0.714286	-0.714286	1.458632	1.458499	0.006589
15	1.000000	-0.714286	1.678578	1.678578	0.000000
16	-1.000000	-0.428571	1.531859	1.531859	0.000000
17	-0.714286	-0.428571	1.312758	1.311779	0.057783
18	-0.428571	-0.428571	1.169185	1.165060	0.301705
19	-0.142857	-0.428571	1.094496	1.091700	0.232258
20	0.142857	-0.428571	1.093755	1.091700	0.170720
21	0.428571	-0.428571	1.169070	1.165060	0.293303
22	0.714286	-0.428571	1.313384	1.311779	0.094727
23	1.000000	-0.428571	1.531859	1.531859	0.000000
24	-1.000000	-0.142857	1.458499	1.458499	0.000000
25	-0.714286	-0.142857	1.239827	1.238419	0.091940
26	-0.428571	-0.142857	1.093490	1.091700	0.148677
27	-0.142857	-0.142857	1.020969	1.018340	0.252638
28	0.142857	-0.142857	1.020902	1.018340	0.246189
29	0.428571	-0.142857	1.094352	1.091700	0.220289
30	0.714286	-0.142857	1.238931	1.238419	0.033398
31	1.000000	-0.142857	1.458499	1.458499	0.000000

Table 3: $t=0.800000$

node	x	y	appro	exact	relative-error(%)
0	-1.000000	-1.000000	1.602388	1.602388	0.000000
1	-0.714286	-1.000000	1.454865	1.454865	0.000000
2	-0.428571	-1.000000	1.356516	1.356516	0.000000
3	-0.142857	-1.000000	1.307341	1.307341	0.000000
4	0.142857	-1.000000	1.307341	1.307341	0.000000
5	0.428571	-1.000000	1.356516	1.356516	0.000000
6	0.714286	-1.000000	1.454865	1.454865	0.000000
7	1.000000	-1.000000	1.602388	1.602388	0.000000
8	-1.000000	-0.714286	1.454865	1.454865	0.000000
9	-0.714286	-0.714286	1.308974	1.307341	0.080834
10	-0.428571	-0.714286	1.210850	1.208992	0.109706
11	-0.142857	-0.714286	1.155936	1.159817	0.253571
12	0.142857	-0.714286	1.167400	1.159817	0.495388
13	0.428571	-0.714286	1.204634	1.208992	0.257294
14	0.714286	-0.714286	1.307845	1.307341	0.024954
15	1.000000	-0.714286	1.454865	1.454865	0.000000
16	-1.000000	-0.428571	1.356516	1.356516	0.000000
17	-0.714286	-0.428571	1.204592	1.208992	0.259764
18	-0.428571	-0.428571	1.118359	1.110643	0.564349
19	-0.142857	-0.428571	1.064378	1.061468	0.241651
20	0.142857	-0.428571	1.057430	1.061468	0.335397
21	0.428571	-0.428571	1.117520	1.110643	0.502931
22	0.714286	-0.428571	1.210198	1.208992	0.071218
23	1.000000	-0.428571	1.356516	1.356516	0.000000
24	-1.000000	-0.142857	1.307341	1.307341	0.000000
25	-0.714286	-0.142857	1.167133	1.159817	0.477981
26	-0.428571	-0.142857	1.055957	1.061468	0.457671
27	-0.142857	-0.142857	1.015636	1.012294	0.321125
28	0.142857	-0.142857	1.015322	1.012294	0.290982
29	0.428571	-0.142857	1.064976	1.061468	0.291324
30	0.714286	-0.142857	1.157674	1.159817	0.140033
31	1.000000	-0.142857	1.307341	1.307341	0.000000

Table 4: $t=1.200000$