MATH 590: Meshfree Methods

Chapter 43: RBF-PS Methods in MATLAB

Greg Fasshauer

Department of Applied Mathematics
Illinois Institute of Technology

Fall 2010

Outline

The coupling of RBF collocation and pseudospectral methods discussed in the previous chapter has provided a number of new insights.

The coupling of RBF collocation and pseudospectral methods discussed in the previous chapter has provided a number of new insights.

- We can apply many standard pseudospectral procedures to RBF solvers.

The coupling of RBF collocation and pseudospectral methods discussed in the previous chapter has provided a number of new insights.

- We can apply many standard pseudospectral procedures to RBF solvers.
- In particular, we now have "standard" procedures for solving time-dependent PDEs with RBFs.

The coupling of RBF collocation and pseudospectral methods discussed in the previous chapter has provided a number of new insights.

- We can apply many standard pseudospectral procedures to RBF solvers.
- In particular, we now have "standard" procedures for solving time-dependent PDEs with RBFs.

In this chapter we illustrate how the RBF pseudospectral approach can be applied in a way very similar to standard polynomial pseudospectral methods.

The coupling of RBF collocation and pseudospectral methods discussed in the previous chapter has provided a number of new insights.

- We can apply many standard pseudospectral procedures to RBF solvers.
- In particular, we now have “standard” procedures for solving time-dependent PDEs with RBFs.

In this chapter we illustrate how the RBF pseudospectral approach can be applied in a way very similar to standard polynomial pseudospectral methods.

Among our numerical illustrations are several examples taken from the book [Trefethen (2000)] (see Programs 17, 35 and 36 there).

The coupling of RBF collocation and pseudospectral methods discussed in the previous chapter has provided a number of new insights.

- We can apply many standard pseudospectral procedures to RBF solvers.
- In particular, we now have “standard” procedures for solving time-dependent PDEs with RBFs.

In this chapter we illustrate how the RBF pseudospectral approach can be applied in a way very similar to standard polynomial pseudospectral methods.

Among our numerical illustrations are several examples taken from the book [Trefethen (2000)] (see Programs 17, 35 and 36 there).

We will also use the 1D transport equation from the previous chapter to compare the RBF and polynomial PS methods.

Outline

How to compute the discretized differential operators

In order to compute, for example, a first-order differentiation matrix we need to remember that — by the chain rule — the derivative of an RBF will be of the general form

$$\frac{\partial}{\partial x}\varphi(\|\boldsymbol{x}\|) = \frac{x}{r}\frac{\mathrm{d}}{\mathrm{d}r}\varphi(r).$$

How to compute the discretized differential operators

In order to compute, for example, a first-order differentiation matrix we need to remember that — by the chain rule — the derivative of an RBF will be of the general form

$$\frac{\partial}{\partial x}\varphi(\|\boldsymbol{x}\|) = \frac{x}{r}\frac{\mathrm{d}}{\mathrm{d}r}\varphi(r).$$

We require both

- the distances, r,
- and differences in x, where x is the first component of $\boldsymbol{x}$.

How to compute the discretized differential operators

In order to compute, for example, a first-order differentiation matrix we need to remember that — by the chain rule — the derivative of an RBF will be of the general form

$$\frac{\partial}{\partial x}\varphi(\|\boldsymbol{x}\|) = \frac{x}{r}\frac{\mathrm{d}}{\mathrm{d}r}\varphi(r).$$

We require both

- the distances, r,
- and differences in x, where x is the first component of $\boldsymbol{x}$.

In our first MATLAB subroutine `DRBF.m` we compute these distance and difference matrices on lines 5 and 6.

How to compute the discretized differential operators

In order to compute, for example, a first-order differentiation matrix we need to remember that — by the chain rule — the derivative of an RBF will be of the general form

$$\frac{\partial}{\partial x}\varphi(\|\boldsymbol{x}\|) = \frac{x}{r}\frac{\mathrm{d}}{\mathrm{d}r}\varphi(r).$$

We require both

- the distances, r,
- and differences in x, where x is the first component of $\boldsymbol{x}$.

In our first MATLAB subroutine `DRBF.m` we compute these distance and difference matrices on lines 5 and 6.
The differentiation matrix is then given by (see lines 8–10)

$$\mathsf{D} = \mathsf{A}_x\mathsf{A}^{-1}.$$

How to compute the discretized differential operators

In order to compute, for example, a first-order differentiation matrix we need to remember that — by the chain rule — the derivative of an RBF will be of the general form

$$\frac{\partial}{\partial x}\varphi(\|\boldsymbol{x}\|) = \frac{x}{r}\frac{\mathrm{d}}{\mathrm{d}r}\varphi(r).$$

We require both

- the distances, r,
- and differences in x, where x is the first component of $\boldsymbol{x}$.

In our first MATLAB subroutine `DRBF.m` we compute these distance and difference matrices on lines 5 and 6.

The differentiation matrix is then given by (see lines 8–10)

$$\mathsf{D} = \mathsf{A}_x\mathsf{A}^{-1}.$$

Note the use of the matrix right division operator `/` or `mrdivide` in MATLAB on line 10 used to solve the system $\mathsf{D}\mathsf{A} = \mathsf{A}_x$ for D.

Program (DRBF.m)

```
function [D,x] = DRBF(N,rbf,dxrbf)
if N==0, D=0; x=1; return, end
x = cos(pi*(0:N)/N)'; x = flipud(x); % Chebyshev pts.
mine = .1; maxe = 10;    % Shape parameter interval
r = DistanceMatrix(x,x);
dx = DifferenceMatrix(x,x);
ep=fminbnd(@(ep) CostEpsilonDRBF(ep,r,dx,rbf,dxrbf),...
                                  mine,maxe);
A = rbf(ep,r);
Ax = dxrbf(ep,r,dx);
D = Ax/A;
```

Program (`DRBF.m`)

```
function [D,x] = DRBF(N,rbf,dxrbf)
if N==0, D=0; x=1; return, end
x = cos(pi*(0:N)/N)'; x = flipud(x); % Chebyshev pts.
mine = .1; maxe = 10;    % Shape parameter interval
r = DistanceMatrix(x,x);
dx = DifferenceMatrix(x,x);
ep=fminbnd(@(ep) CostEpsilonDRBF(ep,r,dx,rbf,dxrbf),...
                                   mine,maxe);
A = rbf(ep,r);
Ax = dxrbf(ep,r,dx);
D = Ax/A;
```

Remark

`DRBF.m` *is a little more complicated than it needs to be since we include an LOOCV-optimization of the RBF shape parameter.*

Program (DRBF.m)

```
function [D,x] = DRBF(N,rbf,dxrbf)
if N==0, D=0; x=1; return, end
x = cos(pi*(0:N)/N)'; x = flipud(x); % Chebyshev pts.
mine = .1; maxe = 10;    % Shape parameter interval
r = DistanceMatrix(x,x);
dx = DifferenceMatrix(x,x);
ep=fminbnd(@(ep) CostEpsilonDRBF(ep,r,dx,rbf,dxrbf),...
                                         mine,maxe);
A = rbf(ep,r);
Ax = dxrbf(ep,r,dx);
D = Ax/A;
```

Remark

DRBF.m *is a little more complicated than it needs to be since we include an LOOCV-optimization of the RBF shape parameter.*

Below we modify the basic routine CostEpsilon.m *so that we optimize* ε *for the matrix problem* $\mathsf{D} = \mathsf{A}_x\mathsf{A}^{-1} \iff \mathsf{A}^T\mathsf{D}^T = (\mathsf{A}_x)^T$.

Program (CostEpsilonDRBF.m)

```
function ceps = CostEpsilonDRBF(ep,r,dx,rbf,dxrbf)
N = size(r,2);
A = rbf(ep,r);    % = A^T since A is symmetric
rhs = dxrbf(ep,r,dx)';    % A_x^T
invA = pinv(A);
EF = (invA*rhs)./repmat(diag(invA),1,N);
ceps = norm(EF(:));
```

Program (CostEpsilonDRBF.m)

```
1   function ceps = CostEpsilonDRBF(ep,r,dx,rbf,dxrbf)
2   N = size(r,2);
3   A = rbf(ep,r);    % = A^T since A is symmetric
4   rhs = dxrbf(ep,r,dx)';    % A_x^T
5   invA = pinv(A);
6   EF = (invA*rhs)./repmat(diag(invA),1,N);
7   ceps = norm(EF(:));
```

Remark

Note that `CostEpsilonDRBF.m` *is very similar to* `CostEpsilon.m`*. Now, however, we compute a right-hand side matrix corresponding to the transpose of* A_x*.*

Therefore, the denominator — which remains the same for all right-hand sides — needs to be cloned on line 6 via the `repmat` *command.*

The cost of ε *is now the Frobenius norm of the matrix* `EF`*.*

We illustrate the use of the subroutine `DBRF.m` by solving a 1-D transport equation.
Consider

$$\begin{aligned} u_t(x,t) + c u_x(x,t) &= 0, \quad x > -1, \; t > 0, \\ u(-1,t) &= 0, \\ u(x,0) &= f(x), \end{aligned}$$

with the well-known solution

$$u(x,t) = f(x - ct).$$

Program (TransportDRBF.m)

```
rbf = @(e,r) exp(-(e*r).^2);      % Gaussian RBF
dxrbf = @(e,r,dx) -2*dx*e^2.*exp(-(e*r).^2);
f = @(x) max(64*(-x).^3.*(1+x).^3,0);
N = 20;  [D,x] = DRBF(N,rbf,dxrbf);
dt = 0.001; t = 0; c = 1; v = f(x);
tmax = 1; tplot = .02; plotgap = round(tplot/dt);
dt = tplot/plotgap; nplots = round(tmax/tplot);
data = [v'; zeros(nplots,N+1)]; tdata = t;
for i = 1:nplots
   for n = 1:plotgap
      t = t+dt;
      vv = v(end-1);
      v = v - dt*c*(D*v);      % explicit Euler
      v(1) = 0; v(end) = vv;
   end
   data(i+1,:) = v'; tdata = [tdata; t];
end
surf(x,tdata,data), view(10,70), colormap('default');
axis([-1 1 0 tmax 0 1]), ylabel t, zlabel u, grid off
xx = linspace(-1,1,101);  vone = f(xx-c);
w = interp1(x,v,xx);
maxErr = norm(w-vone,inf)
```

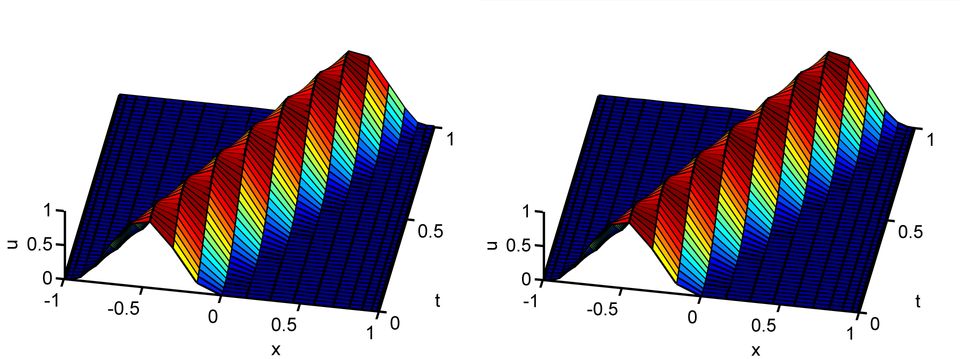

Figure: Time profiles of the solution to the transport equation for $0 \leq t \leq 1$ with initial profile $f(x) = \max(-64x^3(1+x)^3, 0)$ and unit wave speed based on Gaussian RBFs with $\varepsilon = 1.874049$ (left) and Chebyshev PS method (right). Explicit Euler time-stepping with ($\Delta t = 0.001$), and 21 Chebyshev points.

The maximum error for the Gaussian solution at time $t = 1$ is 0.0416, while for the PS solution we get 0.0418.

Remark

- *We could use almost the identical code to solve this problem with a Chebyshev pseudospectral method as discussed in [Trefethen (2000)].*

Remark

- *We could use almost the identical code to solve this problem with a Chebyshev pseudospectral method as discussed in [Trefethen (2000)].*

- *The only difference in the PS-code is the replacement of line 4 in* `TransportDRBF.m` *by*

```
4   N=20; [D,x] = cheb(N); x = flipud(x); D = -D;
```

where `cheb.m` *is the subroutine provided on page 54 of [Trefethen (2000)] for spectral differentiation.*

Remark

- *We could use almost the identical code to solve this problem with a Chebyshev pseudospectral method as discussed in [Trefethen (2000)].*
- *The only difference in the PS-code is the replacement of line 4 in* `TransportDRBF.m` *by*

```
4   N=20; [D,x] = cheb(N); x = flipud(x); D = -D;
```

  where `cheb.m` *is the subroutine provided on page 54 of [Trefethen (2000)] for spectral differentiation.*
- *Note that Trefethen's* `cheb` *is based on a "right-to-left" orientation of the collocation points, and therefore we need to "correct" the points and matrix* `D`.

Outline

We now explain how the Contour-Padé algorithm of [Fornberg and Wright (2004)] can be used to compute RBF differentiation matrices.

We now explain how the Contour-Padé algorithm of [Fornberg and Wright (2004)] can be used to compute RBF differentiation matrices.

In its original form the Contour-Padé algorithm allows us to stably evaluate RBF interpolants based on infinitely smooth RBFs for extreme choices of the shape parameter ε (in particular $\varepsilon \to 0$).

We now explain how the Contour-Padé algorithm of [Fornberg and Wright (2004)] can be used to compute RBF differentiation matrices.

In its original form the Contour-Padé algorithm allows us to stably evaluate RBF interpolants based on infinitely smooth RBFs for extreme choices of the shape parameter ε (in particular $\varepsilon \to 0$).

The Contour-Padé algorithm uses FFTs and Padé approximations to evaluate the function

$$\hat{u}(\boldsymbol{x}, \varepsilon) = \boldsymbol{b}^T(\boldsymbol{x}, \varepsilon)(\mathsf{A}(\varepsilon))^{-1}\boldsymbol{f} \tag{1}$$

with $\boldsymbol{b}(\boldsymbol{x}, \varepsilon)_j = \varphi_\varepsilon(\|\boldsymbol{x} - \boldsymbol{x}_j\|)$ at some evaluation point $\boldsymbol{x}$ and $\mathsf{A}(\varepsilon)_{i,j} = \varphi_\varepsilon(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$.

- If we evaluate $\hat{u}$ at all of the collocation points $\boldsymbol{x}_i$, $i = 1, \ldots, N$, for some fixed value of ε, then $\boldsymbol{b}^T(\boldsymbol{x}, \varepsilon)$ turns into the matrix $\mathsf{A}(\varepsilon)$.

- If we evaluate $\hat{u}$ at all of the collocation points $\boldsymbol{x}_i$, $i = 1, \ldots, N$, for some fixed value of ε, then $\boldsymbol{b}^T(\boldsymbol{x}, \varepsilon)$ turns into the matrix $\mathsf{A}(\varepsilon)$.
- In the case of interpolation this is pointless.

- If we evaluate $\hat{u}$ at all of the collocation points $\boldsymbol{x}_i$, $i = 1, \ldots, N$, for some fixed value of ε, then $\boldsymbol{b}^T(\boldsymbol{x}, \varepsilon)$ turns into the matrix $\mathsf{A}(\varepsilon)$.
- In the case of interpolation this is pointless.
- If the Contour-Padé algorithm is adapted to replace the vector $\boldsymbol{b}^T(\boldsymbol{x}, \varepsilon)$ (corresponding to evaluation at a single point $\boldsymbol{x}$) with the matrix $\mathsf{A}_{\mathcal{L}}$ based on the differential operator (corresponding to evaluation at all collocation points), then

$$\mathsf{A}_{\mathcal{L}}(\varepsilon)(\mathsf{A}(\varepsilon))^{-1}\boldsymbol{u}$$

computes the values of the (spatial) derivative of $\boldsymbol{u}$ on the collocation points $\boldsymbol{x}_i$.

- If we evaluate $\hat{u}$ at all of the collocation points $\boldsymbol{x}_i$, $i = 1, \ldots, N$, for some fixed value of ε, then $\boldsymbol{b}^T(\boldsymbol{x}, \varepsilon)$ turns into the matrix $\mathsf{A}(\varepsilon)$.
- In the case of interpolation this is pointless.
- If the Contour-Padé algorithm is adapted to replace the vector $\boldsymbol{b}^T(\boldsymbol{x}, \varepsilon)$ (corresponding to evaluation at a single point $\boldsymbol{x}$) with the matrix $\mathsf{A}_{\mathcal{L}}$ based on the differential operator (corresponding to evaluation at all collocation points), then

$$\mathsf{A}_{\mathcal{L}}(\varepsilon)(\mathsf{A}(\varepsilon))^{-1}\boldsymbol{u}$$

  computes the values of the (spatial) derivative of $\boldsymbol{u}$ on the collocation points $\boldsymbol{x}_i$.
- Boundary conditions can be incorporated later as in the standard PS approach (see, e.g., [Trefethen (2000)] or Chapter 42).

- This means that we can add another subroutine to compute the differentiation matrix on line 4 of `TransportDRBF.m` via the Contour-Padé algorithm.

- This means that we can add another subroutine to compute the differentiation matrix on line 4 of `TransportDRBF.m` via the Contour-Padé algorithm.
- We compare
  - a solution based on the Contour-Padé algorithm for Gaussian RBFs in the limiting case $\varepsilon \to 0$
  - to the two methods described earlier (based on `DRBF` and `cheb`).

- This means that we can add another subroutine to compute the differentiation matrix on line 4 of `TransportDRBF.m` via the Contour-Padé algorithm.
- We compare
  - a solution based on the Contour-Padé algorithm for Gaussian RBFs in the limiting case $\varepsilon \to 0$
  - to the two methods described earlier (based on `DRBF` and `cheb`).
- All methods use an implicit Euler method with time step $\Delta t = 0.001$ for the time discretization.

- This means that we can add another subroutine to compute the differentiation matrix on line 4 of `TransportDRBF.m` via the Contour-Padé algorithm.
- We compare
  - a solution based on the Contour-Padé algorithm for Gaussian RBFs in the limiting case $\varepsilon \to 0$
  - to the two methods described earlier (based on `DRBF` and `cheb`).
- All methods use an implicit Euler method with time step $\Delta t = 0.001$ for the time discretization.
- For an implicit time-stepping method both the Contour-Padé approach and the `DRBF` approach require an inversion of the differentiation matrix.

- This means that we can add another subroutine to compute the differentiation matrix on line 4 of `TransportDRBF.m` via the Contour-Padé algorithm.
- We compare
  - a solution based on the Contour-Padé algorithm for Gaussian RBFs in the limiting case $\varepsilon \to 0$
  - to the two methods described earlier (based on `DRBF` and `cheb`).
- All methods use an implicit Euler method with time step $\Delta t = 0.001$ for the time discretization.
- For an implicit time-stepping method both the Contour-Padé approach and the `DRBF` approach require an inversion of the differentiation matrix.
- Recall that our theoretical discussion suggested that this is justified as long as we're in the limiting case $\varepsilon \to 0$ and one space dimension.

- This means that we can add another subroutine to compute the differentiation matrix on line 4 of `TransportDRBF.m` via the Contour-Padé algorithm.
- We compare
  - a solution based on the Contour-Padé algorithm for Gaussian RBFs in the limiting case $\varepsilon \to 0$
  - to the two methods described earlier (based on `DRBF` and `cheb`).
- All methods use an implicit Euler method with time step $\Delta t = 0.001$ for the time discretization.
- For an implicit time-stepping method both the Contour-Padé approach and the `DRBF` approach require an inversion of the differentiation matrix.
- Recall that our theoretical discussion suggested that this is justified as long as we're in the limiting case $\varepsilon \to 0$ and one space dimension.
- We will see that the non-limiting case (using `DRBF`) seems to work just as well.

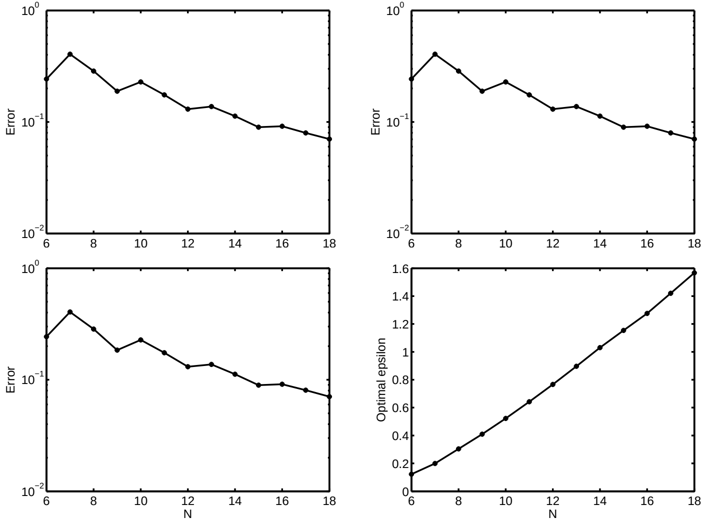

Figure: Errors at $t = 1$ for transport equation. Top: Gaussian RBF with $\varepsilon = 0$ (left) and Chebyshev PS-solution (right). Bottom: Gaussian RBF with "optimal" ε (left) and corresponding ε-values (right). Variable spatial discretization N. Implicit Euler method with $\Delta t = 0.001$.

Remark

- *The "optimal" ε ranges almost linearly increasing from* 0.122661 *at* $N = 6$ *to* 1.566594 *at* $N = 18$.

Remark

- *The "optimal" ε ranges almost linearly increasing from* 0.122661 *at* $N = 6$ *to* 1.566594 *at* $N = 18$.
- *We can see that the errors for all three methods are virtually identical.*

Remark

- *The "optimal" ε ranges almost linearly increasing from* 0.122661 *at* $N = 6$ *to* 1.566594 *at* $N = 18$.
- *We can see that the errors for all three methods are virtually identical.*
- *Unfortunately, in this experiment we are limited to this small range of N since for* $N \geq 19$ *the Contour-Padé solution becomes unreliable.*

Remark

- *The "optimal" ε ranges almost linearly increasing from* 0.122661 *at* $N = 6$ *to* 1.566594 *at* $N = 18$.
- *We can see that the errors for all three methods are virtually identical.*
- *Unfortunately, in this experiment we are limited to this small range of* N *since for* $N \geq 19$ *the Contour-Padé solution becomes unreliable.*
- *The remarkable agreement of all three solutions for these small values of* N *seems to indicate that the errors in the solution are mostly due to the time-stepping method used.*

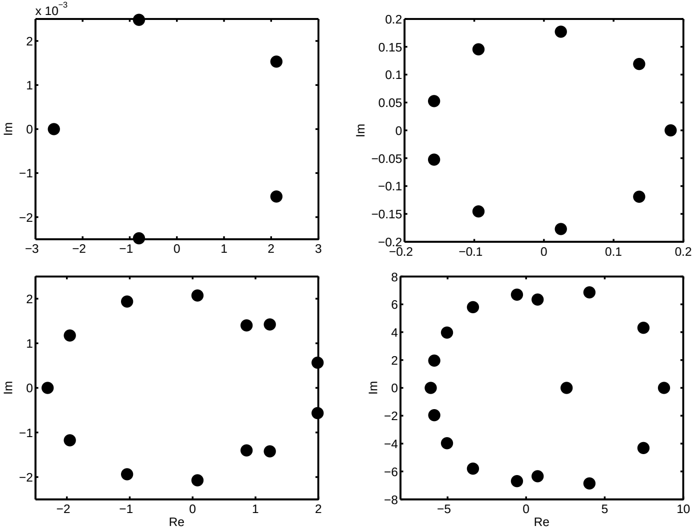

Figure: Spectra of differentiation matrices for Gaussian RBF with $\varepsilon = 0$ on Chebyshev collocation points obtained with the Contour-Padé algorithm and $N = 5, 9, 13, 17$.

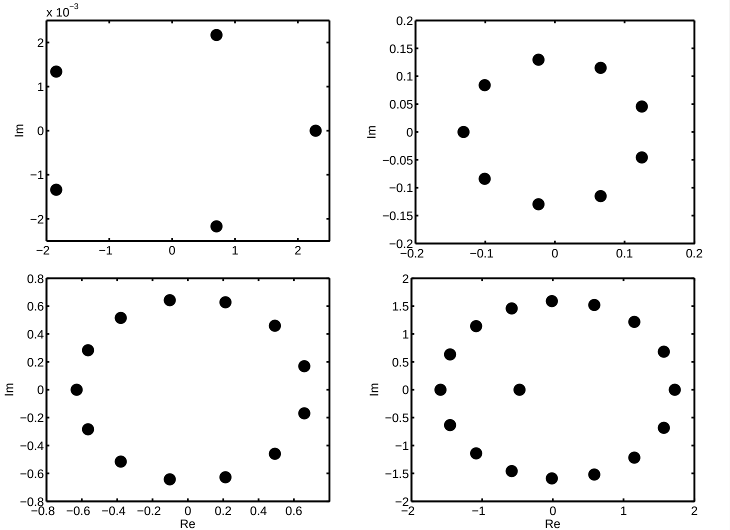

Figure: Spectra of differentiation matrices for Chebyshev pseudospectral method on Chebyshev collocation points with $N = 5, 9, 13, 17$.

Remark

- *The plots for the Gaussian and Chebyshev methods show some similarities, but also some differences.*

Remark

- *The plots for the Gaussian and Chebyshev methods show some similarities, but also some differences.*
- *The general distribution of the eigenvalues for the two methods is quite similar.*

Remark

- *The plots for the Gaussian and Chebyshev methods show some similarities, but also some differences.*
- *The general distribution of the eigenvalues for the two methods is quite similar.*
- *The spectra for the Contour-Padé algorithm with Gaussian RBFs seem to be more or less a slightly stretched reflection about the imaginary axis of the spectra of the Chebyshev pseudospectral method.*

Remark

- *The plots for the Gaussian and Chebyshev methods show some similarities, but also some differences.*
- *The general distribution of the eigenvalues for the two methods is quite similar.*
- *The spectra for the Contour-Padé algorithm with Gaussian RBFs seem to be more or less a slightly stretched reflection about the imaginary axis of the spectra of the Chebyshev pseudospectral method.*
- *The differences increase as N increases.*

Remark

- *The plots for the Gaussian and Chebyshev methods show some similarities, but also some differences.*
- *The general distribution of the eigenvalues for the two methods is quite similar.*
- *The spectra for the Contour-Padé algorithm with Gaussian RBFs seem to be more or less a slightly stretched reflection about the imaginary axis of the spectra of the Chebyshev pseudospectral method.*
- *The differences increase as N increases.*
- *This is not surprising since the Contour-Padé algorithm is known to be unreliable for larger values of N.*

Outline

For polynomial differentiation matrices higher-order derivatives can be computed by repeatedly applying the first-order differentiation matrix, i.e.,

$$\mathsf{D}^{(k)} = \mathsf{D}^k,$$

where D is the standard first-order differentiation matrix and $\mathsf{D}^{(k)}$ is the matrix corresponding to the k-th (univariate) derivative.

For polynomial differentiation matrices higher-order derivatives can be computed by repeatedly applying the first-order differentiation matrix, i.e.,

$$\mathsf{D}^{(k)} = \mathsf{D}^k,$$

where D is the standard first-order differentiation matrix and $\mathsf{D}^{(k)}$ is the matrix corresponding to the k-th (univariate) derivative.

Unfortunately, this does not carry over to the general RBF case (just as is does not hold for periodic Fourier spectral differentiation matrices, either).

For polynomial differentiation matrices higher-order derivatives can be computed by repeatedly applying the first-order differentiation matrix, i.e.,

$$\mathsf{D}^{(k)} = \mathsf{D}^k,$$

where D is the standard first-order differentiation matrix and $\mathsf{D}^{(k)}$ is the matrix corresponding to the k-th (univariate) derivative.

Unfortunately, this does not carry over to the general RBF case (just as is does not hold for periodic Fourier spectral differentiation matrices, either).

We therefore need to provide separate MATLAB code for higher-order differentiation matrices.

Program (D2RBF.m)

```
function [D2,x] = D2RBF(N,rbf,d2rbf)
if N==0, D2=0; x=1; return, end
x = cos(pi*(0:N)/N)';     % Chebyshev points
mine = .1; maxe = 10;     % Shape parameter interval
r = DistanceMatrix(x,x);
ep=fminbnd(@(ep) CostEpsilonD2RBF(ep,r,rbf,d2rbf),...
                                             mine,maxe);
A = rbf(ep,r);
AD2 = d2rbf(ep,r);
D2 = AD2/A;
```

The only new thing that is needed for D2RBFS is the appropriate formula for the derivative of the RBF passed to `D2RBF` via the parameter `d2rbf`.

Remark

- *We do not list* `CostEpsilonD2RBF`*.*

Remark

- *We do not list* `CostEpsilonD2RBF`*.*
- *It differs from* `CostEpsilonDRBF` *only in the definition of the right-hand side matrix which now becomes*

```
rhs = d2rbf(ep,r)';
```

Remark

- *We do not list* `CostEpsilonD2RBF`*.*
- *It differs from* `CostEpsilonDRBF` *only in the definition of the right-hand side matrix which now becomes*

```
rhs = d2rbf(ep,r)';
```

- *Also, the number and type of parameters that are passed to the functions are different since the first-order derivative requires differences of collocation points and the second-order derivative does not.*

We illustrate the use of the subroutine `D2RBF.m` with a modification of Program 35 in [Trefethen (2000)] which is concerned with the solution of the nonlinear reaction-diffusion (or Allen-Cahn) equation.

We illustrate the use of the subroutine `D2RBF.m` with a modification of Program 35 in [Trefethen (2000)] which is concerned with the solution of the nonlinear reaction-diffusion (or Allen-Cahn) equation.

Consider

$$u_t = \mu u_{xx} + u - u^3, \qquad x \in (-1,1),\ t \geq 0,$$

with parameter μ, initial condition

$$u(x,0) = 0.53x + 0.47\sin\left(-\frac{3}{2}\pi x\right), \qquad x \in [-1,1],$$

and non-homogeneous (time-dependent) boundary conditions

$$\begin{aligned} u(-1,t) &= -1 \\ u(1,t) &= \sin^2(t/5). \end{aligned}$$

We illustrate the use of the subroutine `D2RBF.m` with a modification of Program 35 in [Trefethen (2000)] which is concerned with the solution of the nonlinear reaction-diffusion (or Allen-Cahn) equation.
Consider

$$u_t = \mu u_{xx} + u - u^3, \qquad x \in (-1, 1),\ t \geq 0,$$

with parameter μ, initial condition

$$u(x, 0) = 0.53x + 0.47 \sin\left(-\frac{3}{2}\pi x\right), \qquad x \in [-1, 1],$$

and non-homogeneous (time-dependent) boundary conditions

$$\begin{aligned} u(-1, t) &= -1 \\ u(1, t) &= \sin^2(t/5). \end{aligned}$$

The solution has three steady states ($u = -1, 0, 1$) with the two nonzero states being stable.

We illustrate the use of the subroutine `D2RBF.m` with a modification of Program 35 in [Trefethen (2000)] which is concerned with the solution of the nonlinear reaction-diffusion (or Allen-Cahn) equation.
Consider

$$u_t = \mu u_{xx} + u - u^3, \qquad x \in (-1,1),\ t \geq 0,$$

with parameter μ, initial condition

$$u(x,0) = 0.53x + 0.47 \sin\left(-\frac{3}{2}\pi x\right), \qquad x \in [-1,1],$$

and non-homogeneous (time-dependent) boundary conditions

$$\begin{aligned} u(-1,t) &= -1 \\ u(1,t) &= \sin^2(t/5). \end{aligned}$$

The solution has three steady states ($u = -1, 0, 1$) with the two nonzero states being stable.
The transition between these states is governed by the parameter μ.

We illustrate the use of the subroutine `D2RBF.m` with a modification of Program 35 in [Trefethen (2000)] which is concerned with the solution of the nonlinear reaction-diffusion (or Allen-Cahn) equation.
Consider

$$u_t = \mu u_{xx} + u - u^3, \qquad x \in (-1,1),\ t \geq 0,$$

with parameter μ, initial condition

$$u(x,0) = 0.53x + 0.47 \sin\left(-\frac{3}{2}\pi x\right), \qquad x \in [-1,1],$$

and non-homogeneous (time-dependent) boundary conditions

$$\begin{aligned} u(-1,t) &= -1 \\ u(1,t) &= \sin^2(t/5). \end{aligned}$$

The solution has three steady states ($u = -1, 0, 1$) with the two nonzero states being stable.
The transition between these states is governed by the parameter μ.
Below we use $\mu = 0.01$, and the unstable state should vanish around $t = 30$.

Program (Modification of Program 35 of [Trefethen (2000)])

```
rbf = @(e,r) exp(-e*r).*(15+15*e*r+6*(e*r).^2+(e*r).^3);
d2rbf = @(e,r) e^2*((e*r).^3-3*e*r-3).*exp(-e*r);
N = 20;  [D2,x] = D2RBF(N,rbf,d2rbf);
% Here is the rest of Trefethen's code.
mu = 0.01; dt = min([.01,50*N^(-4)/mu]);
t = 0; v = .53*x + .47*sin(-1.5*pi*x);
tmax = 100; tplot = 2; nplots = round(tmax/tplot);
plotgap = round(tplot/dt); dt = tplot/plotgap;
xx = -1:.025:1; vv = polyval(polyfit(x,v,N),xx);
plotdata = [vv; zeros(nplots,length(xx))]; tdata = t;
for i = 1:nplots
   for n = 1:plotgap
      t = t+dt; v = v + dt*(mu*D2*v + v - v.^3); % Euler
      v(1) = 1 + sin(t/5)^2; v(end) = -1; % BC
   end
   vv = polyval(polyfit(x,v,N),xx);
   plotdata(i+1,:) = vv; tdata = [tdata; t];
end
surf(xx,tdata,plotdata), grid on
axis([-1 1 0 tmax -1 2]), view(-40,55)
colormap('default'); xlabel x, ylabel t, zlabel u
```

Remark

- *Note how easily the nonlinearity is dealt with by incorporating it into the time-stepping method on line 12.*

Remark

- *Note how easily the nonlinearity is dealt with by incorporating it into the time-stepping method on line 12.*
- *The original program in [Trefethen (2000)] is obtained by deleting lines 1–2 and replacing line 3 by a call to* `cheb.m` *followed by the statement* `D2 = D^2`.

Remark

- *Note how easily the nonlinearity is dealt with by incorporating it into the time-stepping method on line 12.*
- *The original program in [Trefethen (2000)] is obtained by deleting lines 1–2 and replacing line 3 by a call to* `cheb.m` *followed by the statement* `D2 = D^2`.
- *In our RBF-PS implementation the majority of the matrix computations are required only once outside the time-stepping procedure when computing the derivative matrix.*

Remark

- *Note how easily the nonlinearity is dealt with by incorporating it into the time-stepping method on line 12.*
- *The original program in [Trefethen (2000)] is obtained by deleting lines 1–2 and replacing line 3 by a call to* `cheb.m` *followed by the statement* `D2 = D^2`*.*
- *In our RBF-PS implementation the majority of the matrix computations are required only once outside the time-stepping procedure when computing the derivative matrix.*
- *Inside the time-stepping loop (on line 12) we require only matrix-vector multiplication.*

Remark

- *Note how easily the nonlinearity is dealt with by incorporating it into the time-stepping method on line 12.*
- *The original program in [Trefethen (2000)] is obtained by deleting lines 1–2 and replacing line 3 by a call to* `cheb.m` *followed by the statement* `D2 = D^2`.
- *In our RBF-PS implementation the majority of the matrix computations are required only once outside the time-stepping procedure when computing the derivative matrix.*
- *Inside the time-stepping loop (on line 12) we require only matrix-vector multiplication.*
- *We point out that this approach is much more efficient than computation of RBF expansion coefficients at every time step (as suggested, e.g., in [Hon and Mao (1999)]).*

Remark

- *Note how easily the nonlinearity is dealt with by incorporating it into the time-stepping method on line 12.*
- *The original program in [Trefethen (2000)] is obtained by deleting lines 1–2 and replacing line 3 by a call to* `cheb.m` *followed by the statement* `D2 = D^2`.
- *In our RBF-PS implementation the majority of the matrix computations are required only once outside the time-stepping procedure when computing the derivative matrix.*
- *Inside the time-stepping loop (on line 12) we require only matrix-vector multiplication.*
- *We point out that this approach is much more efficient than computation of RBF expansion coefficients at every time step (as suggested, e.g., in [Hon and Mao (1999)]).*
- *In fact, this is the main difference between the RBF-PS approach and the collocation approach of Chapters 38–40.*

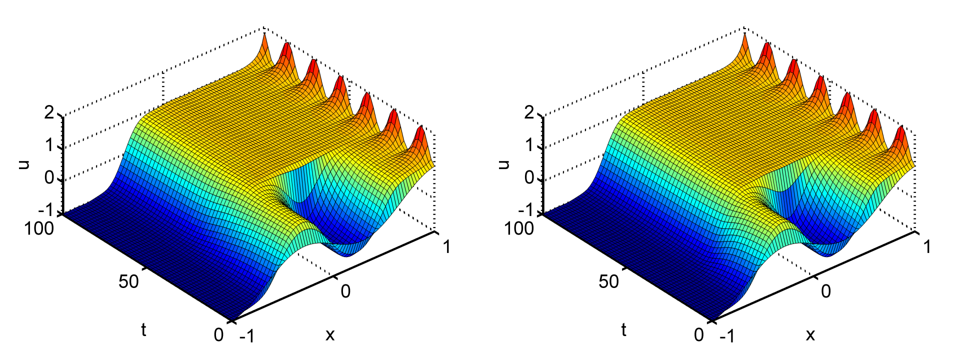

Figure: Solution of the Allen-Cahn equation using the Chebyshev PS-method (left) and an RBF-PS method with cubic Matérn functions $\varphi(r) = (15 + 15\varepsilon r + 6(\varepsilon r)^2 + (\varepsilon r)^3)e^{-\varepsilon r}$ with "optimal" shape parameter $\varepsilon = 0.350952$ (right) with $N = 20$.

Remark

- *We can see that the solution based on Chebyshev polynomials appears to be slightly more accurate since the transition occurs at a slightly later and correct time (i.e., at $t \approx 30$) and is also a little "sharper".*

Remark

- *We can see that the solution based on Chebyshev polynomials appears to be slightly more accurate since the transition occurs at a slightly later and correct time (i.e., at $t \approx 30$) and is also a little "sharper".*

- *The differentiation matrix RBF is obtained directly with* `D2RBF.m` *(i.e., without the Contour-Padé algorithm – since that method is not reliable for 21 points).*

Remark

- *We can see that the solution based on Chebyshev polynomials appears to be slightly more accurate since the transition occurs at a slightly later and correct time (i.e., at $t \approx 30$) and is also a little "sharper".*
- *The differentiation matrix RBF is obtained directly with* `D2RBF.m` *(i.e., without the Contour-Padé algorithm – since that method is not reliable for 21 points).*
- *The plots show that reasonable solutions can also be obtained via this direct (and much simpler) RBF approach.*

Remark

- *We can see that the solution based on Chebyshev polynomials appears to be slightly more accurate since the transition occurs at a slightly later and correct time (i.e., at t ≈ 30) and is also a little "sharper".*
- *The differentiation matrix RBF is obtained directly with* `D2RBF.m` *(i.e., without the Contour-Padé algorithm – since that method is not reliable for 21 points).*
- *The plots show that reasonable solutions can also be obtained via this direct (and much simpler) RBF approach.*
- *True spectral accuracy will no longer be given if $\varepsilon > 0$.*

Outline

Consider the 2D Helmholtz equation (see Program 17 in [Trefethen (2000)])

$$u_{xx} + u_{yy} + k^2 u = f(x, y), \quad x, y \in (-1, 1)^2,$$

with boundary condition

$$u = 0$$

Consider the 2D Helmholtz equation (see Program 17 in [Trefethen (2000)])

$$u_{xx} + u_{yy} + k^2 u = f(x, y), \quad x, y \in (-1, 1)^2,$$

with boundary condition

$$u = 0$$

and exact solution

$$f(x, y) = \exp\left(-10\left[(y-1)^2 + (x - \frac{1}{2})^2\right]\right).$$

Remark

- *To solve this type of (elliptic) problem we need to assume invertibility of the differentiation matrix (even though this may be theoretically questionable).*

Remark

- *To solve this type of (elliptic) problem we need to assume invertibility of the differentiation matrix (even though this may be theoretically questionable).*
- *We compare*
  - *a non-symmetric RBF pseudospectral method*
  - *with a Chebyshev pseudospectral method.*

Remark

- *To solve this type of (elliptic) problem we need to assume invertibility of the differentiation matrix (even though this may be theoretically questionable).*
- *We compare*
    - *a non-symmetric RBF pseudospectral method*
    - *with a Chebyshev pseudospectral method.*
- *We attempt to solve the problem with radial basis functions in two different ways.*

Approach 1:

We apply the same tensor-product technique as in [Trefethen (2000)] using the `kron` function to express the disretized Laplacian on a tensor-product grid of $(N+1) \times (N+1)$ points as

$$\mathsf{L} = \mathsf{I} \otimes \mathsf{D2} + \mathsf{D2} \otimes \mathsf{I}, \tag{2}$$

where

- D2: is the (univariate) second-order differentiation matrix,
- I: is an identity matrix of size $(N+1) \times (N+1)$, and
- $\otimes$: denotes the Kronecker tensor-product.

Approach 1:

We apply the same tensor-product technique as in [Trefethen (2000)] using the `kron` function to express the disretized Laplacian on a tensor-product grid of $(N+1) \times (N+1)$ points as

$$\mathsf{L} = \mathsf{I} \otimes \mathsf{D2} + \mathsf{D2} \otimes \mathsf{I}, \tag{2}$$

where

- D2: is the (univariate) second-order differentiation matrix,
- I: is an identity matrix of size $(N+1) \times (N+1)$, and
- $\otimes$: denotes the Kronecker tensor-product.

- For polynomial PS methods we have $\mathsf{D2} = \mathsf{D}^2$.

Approach 1:

We apply the same tensor-product technique as in [Trefethen (2000)] using the `kron` function to express the disretized Laplacian on a tensor-product grid of $(N+1) \times (N+1)$ points as

$$\mathsf{L} = \mathsf{I} \otimes \mathsf{D2} + \mathsf{D2} \otimes \mathsf{I}, \qquad (2)$$

where

- D2: is the (univariate) second-order differentiation matrix,
- I: is an identity matrix of size $(N+1) \times (N+1)$, and
- $\otimes$: denotes the Kronecker tensor-product.

- For polynomial PS methods we have $\mathsf{D2} = \mathsf{D}^2$.
- For RBFs $\mathsf{D}^2 \neq \mathsf{D}^{(2)}$, and we generate D2 with `D2RBF`.

Approach 1:

We apply the same tensor-product technique as in [Trefethen (2000)] using the `kron` function to express the disretized Laplacian on a tensor-product grid of $(N+1) \times (N+1)$ points as

$$\mathsf{L} = \mathsf{I} \otimes \mathsf{D2} + \mathsf{D2} \otimes \mathsf{I}, \qquad (2)$$

where

- D2: is the (univariate) second-order differentiation matrix,
- I: is an identity matrix of size $(N+1) \times (N+1)$, and
- $\otimes$: denotes the Kronecker tensor-product.

- For polynomial PS methods we have $\mathsf{D2} = \mathsf{D}^2$.
- For RBFs $\mathsf{D}^2 \neq \mathsf{D}^{(2)}$, and we generate D2 with `D2RBF`.
- However, as long as we use tensor-product collocation points and the RBF is separable (such as a Gaussian or a polynomial), we can still use the Kronecker tensor-product construction (2).

Program (Modification of Program 17 of [Trefethen (2000)])

```
rbf = @(e,r) exp(-(e*r).^2);
d2rbf = @(e,r) 2*e^2*(2*(e*r).^2-1).*exp(-(e*r).^2);
N = 24;  [D2,x] = D2RBF(N,rbf,d2rbf); y = x;
[xx,yy] = meshgrid(x,y);  xx = xx(:); yy = yy(:);
I = eye(N+1);
k = 9;
L = kron(I,D2) + kron(D2,I) + k^2*eye((N+1)^2);
b = find(abs(xx)==1 | abs(yy)==1);   % boundary pts
L(b,:) = zeros(4*N,(N+1)^2); L(b,b) = eye(4*N);
f = exp(-10*((yy-1).^2+(xx-.5).^2));
f(b) = zeros(4*N,1);
u = L\f;
uu = reshape(u,N+1,N+1);
[xx,yy] = meshgrid(x,y);
[xxx,yyy] = meshgrid(-1:.0333:1,-1:.0333:1);
uuu = interp2(xx,yy,uu,xxx,yyy,'cubic');
figure, clf, surf(xxx,yyy,uuu),
xlabel x, ylabel y, zlabel u
text(.2,1,.022,sprintf('u(0,0)=%13.11f',uu(N/2+1,N/2+1))
```

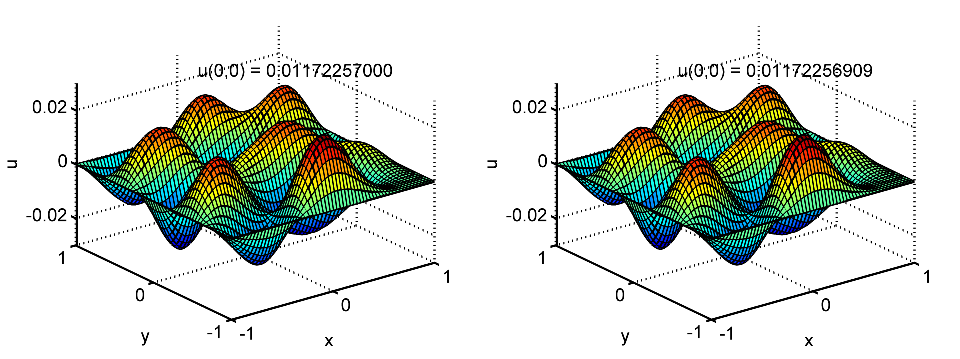

Figure: Solution of the 2D Helmholtz equation with $N = 24$ using the Chebyshev pseudospectral method (left) and Gaussians with $\varepsilon = 2.549845$ (right).

Approach 2:

- This approach allows the use of non-tensor product collocation grids.

Approach 2:

- This approach allows the use of non-tensor product collocation grids.
- We use a direct implementation of the Laplacian of the RBFs.

Approach 2:

- This approach allows the use of non-tensor product collocation grids.
- We use a direct implementation of the Laplacian of the RBFs.
- The only advantage of doing this on a tensor-product grid is that now all RBFs can be used.

Approach 2:

- This approach allows the use of non-tensor product collocation grids.
- We use a direct implementation of the Laplacian of the RBFs.
- The only advantage of doing this on a tensor-product grid is that now all RBFs can be used.
- This approach takes considerably longer to execute since the differentiation matrix is now computed with matrices of size 625×625 instead of the 25×25 univariate differentiation matrix D2 used before.

Approach 2:

- This approach allows the use of non-tensor product collocation grids.
- We use a direct implementation of the Laplacian of the RBFs.
- The only advantage of doing this on a tensor-product grid is that now all RBFs can be used.
- This approach takes considerably longer to execute since the differentiation matrix is now computed with matrices of size 625×625 instead of the 25×25 univariate differentiation matrix D2 used before.
- Moreover, the results are likely to be less accurate since the larger matrices are more prone to ill-conditioning.

Approach 2:

- This approach allows the use of non-tensor product collocation grids.
- We use a direct implementation of the Laplacian of the RBFs.
- The only advantage of doing this on a tensor-product grid is that now all RBFs can be used.
- This approach takes considerably longer to execute since the differentiation matrix is now computed with matrices of size 625×625 instead of the 25×25 univariate differentiation matrix D2 used before.
- Moreover, the results are likely to be less accurate since the larger matrices are more prone to ill-conditioning.
- However, the advantage of this approach is that it frees us of the limitation of polynomial PS methods to tensor-product collocation grids.

Program (Modification II of Program 17 of [Trefethen (2000)])

```
rbf=@(e,r) max(1-e*r,0).^8.*(32*(e*r).^3+25*(e*r).^2+8*
Lrbf = @(e,r) 44*e^2*max(1-e*r,0).^6.*...
                   (88*(e*r).^3+3*(e*r).^2-6*e*r-1);
N = 24; [L,x,y] = LRBF(N,rbf,Lrbf);
[xx,yy] = meshgrid(x,y);
xx = xx(:); yy = yy(:);
k = 9;
L = L + k^2*eye((N+1)^2);
b = find(abs(xx)==1 | abs(yy)==1);      % boundary pts
L(b,:) = zeros(4*N,(N+1)^2); L(b,b) = eye(4*N);
f = exp(-10*((yy-1).^2+(xx-.5).^2));
f(b) = zeros(4*N,1);
u = L\f;
uu = reshape(u,N+1,N+1);
[xx,yy] = meshgrid(x,y);
[xxx,yyy] = meshgrid(-1:.0333:1,-1:.0333:1);
uuu = interp2(xx,yy,uu,xxx,yyy,'cubic');
figure, clf, surf(xxx,yyy,uuu),
xlabel x, ylabel y, zlabel u
text(.2,1,.022,sprintf('u(0,0)=%13.11f',uu(N/2+1,N/2+1)
```

Program (LRBF.m)

```
function [L,x,y] = LRBF(N,rbf,Lrbf)
if N==0, L=0; x=1; return, end
x = cos(pi*(0:N)/N)';    % Chebyshev points
y = x; [xx,yy] = meshgrid(x,y);
% Stretch 2D grids to 1D vectors and put in one array
points = [xx(:) yy(:)];
mine = .1; maxe = 10;    % Shape parameter interval
r = DistanceMatrix(points,points);
ep = fminbnd(@(ep) CostEpsilonLRBF(ep,r,rbf,Lrbf),...
                                  mine,maxe);
fprintf('Using epsilon = %f\n', ep)
A = rbf(ep,r);
AL = Lrbf(ep,r);
L = AL/A;
```

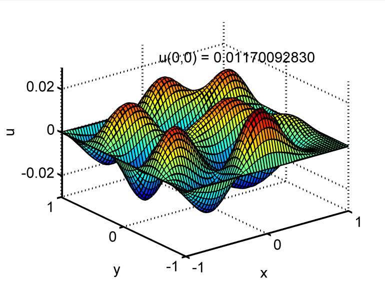

Figure: Solution of the 2D Helmholtz equation using a direct implementation of the Laplacian based on $\varphi_{3,3}(r) = (1-\varepsilon r)_+^8(32(\varepsilon r)^3 + 25(\varepsilon r)^2 + 8\varepsilon r + 1)$ with $\varepsilon = 0.129444$ on 625 tensor-product Chebyshev points.

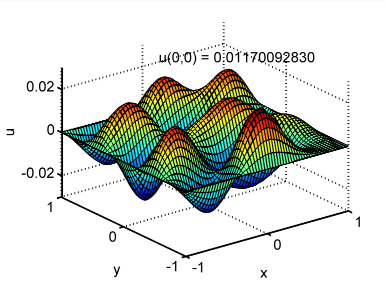

Figure: Solution of the 2D Helmholtz equation using a direct implementation of the Laplacian based on $\varphi_{3,3}(r) = (1-\varepsilon r)_+^8(32(\varepsilon r)^3 + 25(\varepsilon r)^2 + 8\varepsilon r + 1)$ with $\varepsilon = 0.129444$ on 625 tensor-product Chebyshev points.

Remark

- *We use compactly supported Wendland functions in "global mode".*

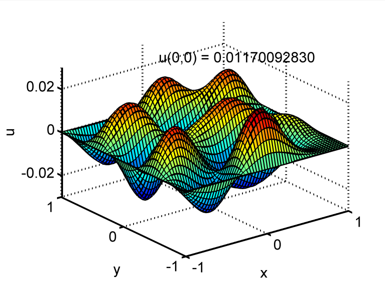

Figure: Solution of the 2D Helmholtz equation using a direct implementation of the Laplacian based on $\varphi_{3,3}(r) = (1-\varepsilon r)_+^8(32(\varepsilon r)^3 + 25(\varepsilon r)^2 + 8\varepsilon r + 1)$ with $\varepsilon = 0.129444$ on 625 tensor-product Chebyshev points.

Remark

- *We use compactly supported Wendland functions in "global mode".*
- *This explains the definition of the basic function in the* MATLAB *code as needed for* `DistanceMatrix.m` *in* `LRBF.m`*.*

Outline

Consider the 2D Laplace equation (see Program 36 of [Trefethen (2000)] and earlier examples)

$$u_{xx} + u_{yy} = 0, \quad x, y \in (-1, 1)^2,$$

with boundary conditions

$$u(x,y) = \begin{cases} \sin^4(\pi x), & y = 1 \text{ and } -1 < x < 0, \\ \frac{1}{5}\sin(3\pi y), & x = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Consider the 2D Laplace equation (see Program 36 of [Trefethen (2000)] and earlier examples)

$$u_{xx} + u_{yy} = 0, \quad x, y \in (-1, 1)^2,$$

with boundary conditions

$$u(x, y) = \begin{cases} \sin^4(\pi x), & y = 1 \text{ and } -1 < x < 0, \\ \frac{1}{5}\sin(3\pi y), & x = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Remark

We don't list the code since it is too similar to previous examples and the original code in [Trefethen (2000)].

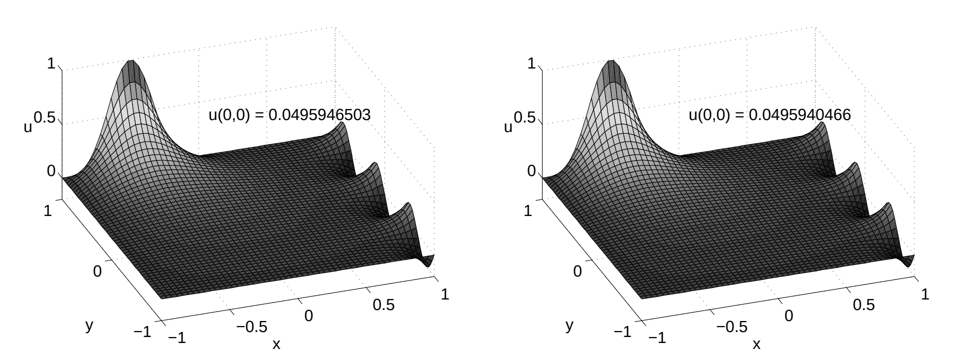

Figure: Solution of the 2D Laplace equation using a Chebyshev PS approach (left) and Gaussian RBFs (right) with $\varepsilon = 2.549845$ on 625 tensor-product Chebyshev collocation points.

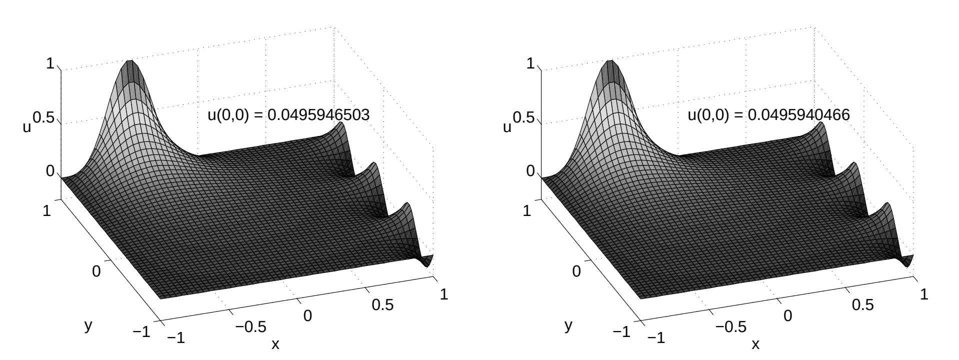

Figure: Solution of the 2D Laplace equation using a Chebyshev PS approach (left) and Gaussian RBFs (right) with $\varepsilon = 2.549845$ on 625 tensor-product Chebyshev collocation points.

The differentiation matrix for the RBF-PS approach is computed using the `D2RBF` and `kron` construction.

Outline

Remark

- *While there is no advantage in going to arbitrarily spaced irregular collocation points for any of the problems presented here, there is nothing that prevents us from doing so for the RBF-PS approach.*

Remark

- *While there is no advantage in going to arbitrarily spaced irregular collocation points for any of the problems presented here, there is nothing that prevents us from doing so for the RBF-PS approach.*
- *A potential advantage of the RBF-PS approach over the standard polynomial methods is the fact that we are not limited to using tensor product grids for higher-dimensional spatial discretizations.*

Remark

- *While there is no advantage in going to arbitrarily spaced irregular collocation points for any of the problems presented here, there is nothing that prevents us from doing so for the RBF-PS approach.*
- *A potential advantage of the RBF-PS approach over the standard polynomial methods is the fact that we are not limited to using tensor product grids for higher-dimensional spatial discretizations.*
- *More applications of the RBF-PS method can be found in [Ferreira and Fasshauer (2006), Ferreira and Fasshauer (2007)].*

Remark

- *While there is no advantage in going to arbitrarily spaced irregular collocation points for any of the problems presented here, there is nothing that prevents us from doing so for the RBF-PS approach.*
- *A potential advantage of the RBF-PS approach over the standard polynomial methods is the fact that we are not limited to using tensor product grids for higher-dimensional spatial discretizations.*
- *More applications of the RBF-PS method can be found in [Ferreira and Fasshauer (2006), Ferreira and Fasshauer (2007)].*
- *Future challenges include*
  - *dealing with larger problems in an efficient and stable way, and*
  - *coming up with preconditioning and FFT-type algorithms.*

Remark

- *While there is no advantage in going to arbitrarily spaced irregular collocation points for any of the problems presented here, there is nothing that prevents us from doing so for the RBF-PS approach.*
- *A potential advantage of the RBF-PS approach over the standard polynomial methods is the fact that we are not limited to using tensor product grids for higher-dimensional spatial discretizations.*
- *More applications of the RBF-PS method can be found in [Ferreira and Fasshauer (2006), Ferreira and Fasshauer (2007)].*
- *Future challenges include*
  - *dealing with larger problems in an efficient and stable way, and*
  - *coming up with preconditioning and FFT-type algorithms.*
- *Eigenvalue stability of RBF-PS methods have been reported in [Platte and Driscoll (2006)].*

References I

- Buhmann, M. D. (2003).
  Radial Basis Functions: Theory and Implementations.
  Cambridge University Press.
- Fasshauer, G. E. (2007).
  Meshfree Approximation Methods with MATLAB.
  World Scientific Publishers.
- Higham, D. J. and Higham, N. J. (2005).
  MATLAB *Guide*.
  SIAM (2nd ed.), Philadelphia.
- Iske, A. (2004).
  Multiresolution Methods in Scattered Data Modelling.
  Lecture Notes in Computational Science and Engineering 37, Springer Verlag (Berlin).
- Trefethen, L. N. (2000).
  Spectral Methods in MATLAB.
  SIAM (Philadelphia, PA).

References II

G. Wahba (1990).
Spline Models for Observational Data.
CBMS-NSF Regional Conference Series in Applied Mathematics 59, SIAM (Philadelphia).

Wendland, H. (2005a).
Scattered Data Approximation.
Cambridge University Press (Cambridge).

Ferreira, A. J. M. and Fasshauer, G. E. (2006)
Computation of natural frequencies of shear deformable beams and plates by an RBF-pseudospectral method.
Comput. Meth. Appl. Mech. Engng. **196**, 134–146.

Ferreira, A. J. M. and Fasshauer, G. E. (2007)
Analysis of natural frequencies of composite plates by an RBF-pseudospectral method.
Composite Structures **79**, pp. 202–210.

References III

Fornberg, B. and Wright, G. (2004).
Stable computation of multiquadric interpolants for all values of the shape parameter.
Comput. Math. Appl. **47**, pp. 497–523.

Hon, Y. C. and Mao, X. Z. (1999).
A radial basis function method for solving options pricing model.
Financial Engineering **8**, pp. 31–49.

Platte, R. B. and Driscoll, T. A. (2006).
Eigenvalue stability of radial basis function discretizations for time-dependent problems.
Comput. Math. Appl. **51** 8, pp. 1251–1268.

Rippa, S. (1999).
An algorithm for selecting a good value for the parameter c in radial basis function interpolation.
Adv. in Comput. Math. **11**, pp. 193–210.