Space-Time Localized Radial Basis Function Collocation Methods for PDEs

Alfa Heryudono

Department of Mathematics
University of Massachusetts Dartmouth

ICERM Topical Workshop
Providence
August 2017

This research is partly supported by NSF DMS-1552238. Joint work with grad student: Jacob Sousa.
Computations are mostly done on UMassD Rapid Prototyping Server.

Dealing with Time-Dependent PDEs for RBF Methods

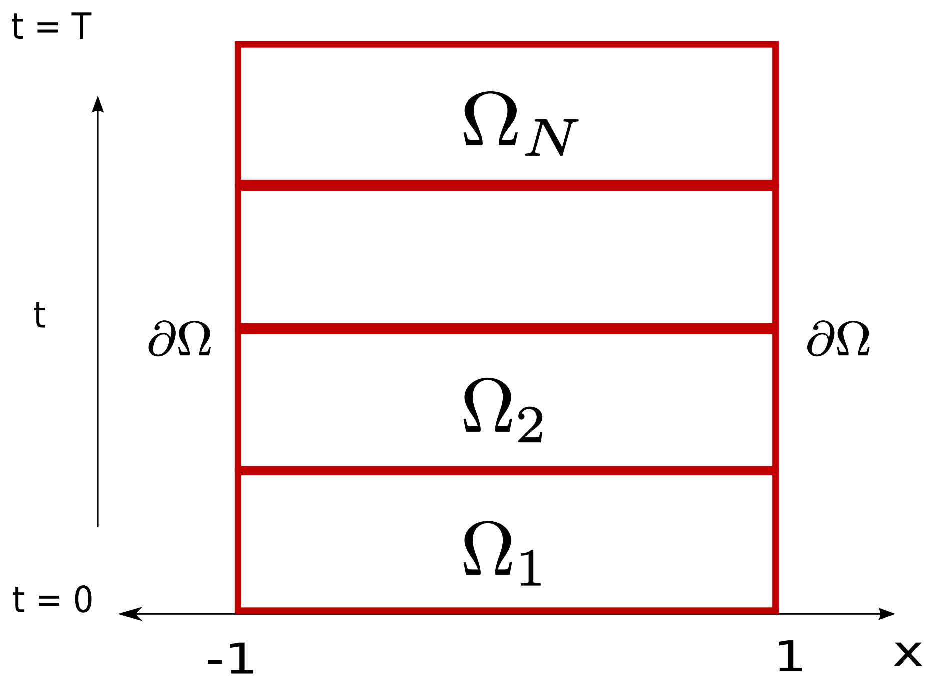
Method of Lines

- RBF discretization in space + common ODE solver in time.
- Min changes of PS/FD codes: replace differentiation matrices with RBF versions (Global, RBF-FD, RBF-PU, etc).
- PS/FD treatments for BCs: Strip-rows, Strip-rows move over columns, fictitious pts/ rect projection (for multiple bcs), penalty, etc.
- Stability for linear pde case: Eigenvalue and Pseudospectra.

Simultaneous Space-Time RBF

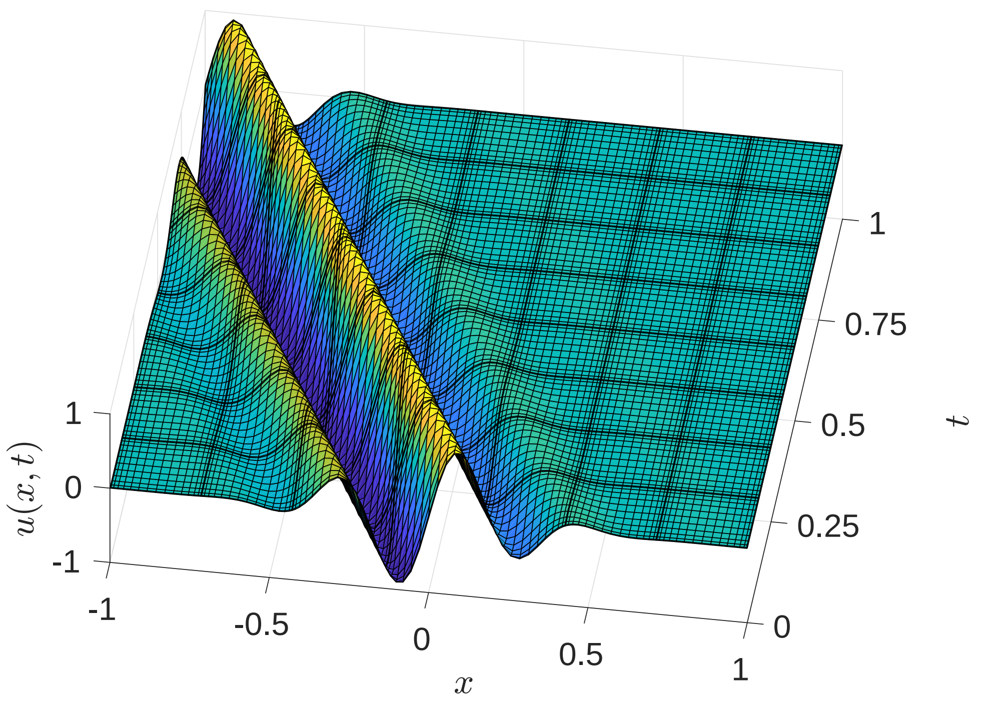
- Boundary value collocation problem in space-time domain. Time is treated as another space variable. RBF-BVP solver have been studied for quite a while.
- Less worry about choosing ODE solver based on PDE types.
- Adaptivity, moving boundary, and BCs: same treatments as in BVP cases.
- No need to rewrite the pde due to var trans (e.g in moving boundary case).
- Analyzing stability is not clear (e.g. in moving boundary case).
- Might be expensive to solve (e.g. finding preconditioner, non-linear case).

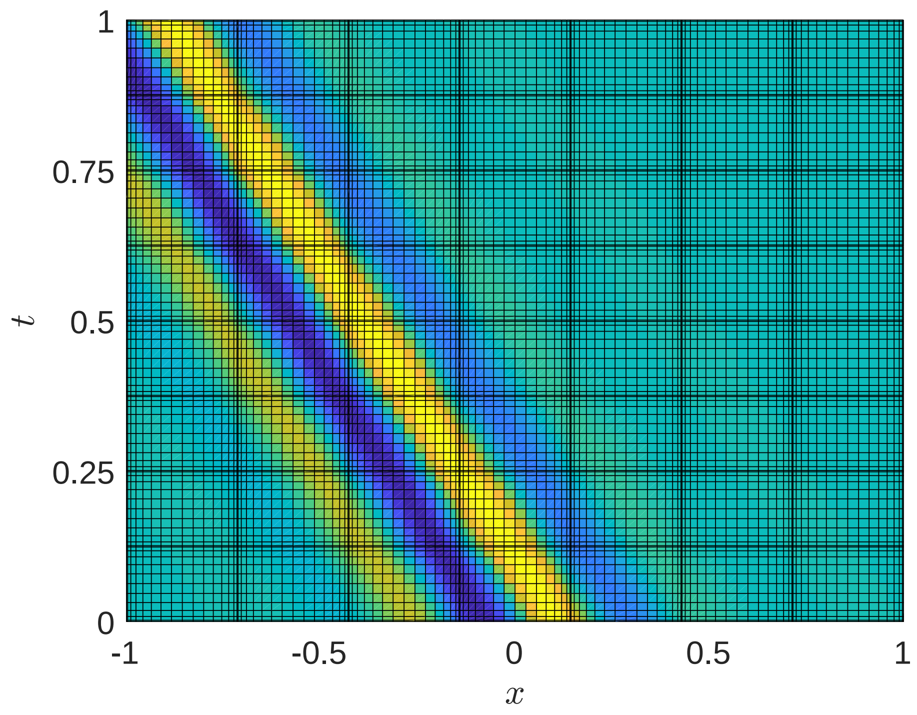
Space-Time PS Collocation Method: 1D+t linear case



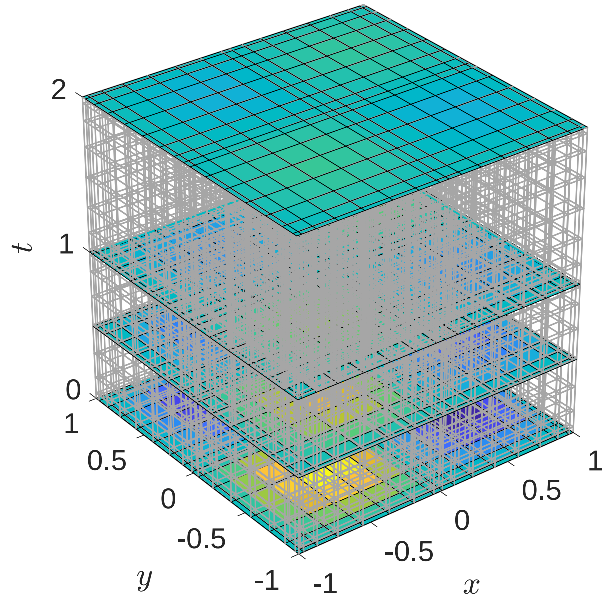
$$\begin{aligned} PDE: &\quad u_t = u_x \\ &\quad (x,t) \in [-1,1) \times (0,T] \\ IC: &\quad u(x,0) = f(x) \\ BC: &\quad u(1,t) = g(t) \end{aligned}$$

Use PS or Block PS (Driscoll-Fornberg) to create differentiation matrices.



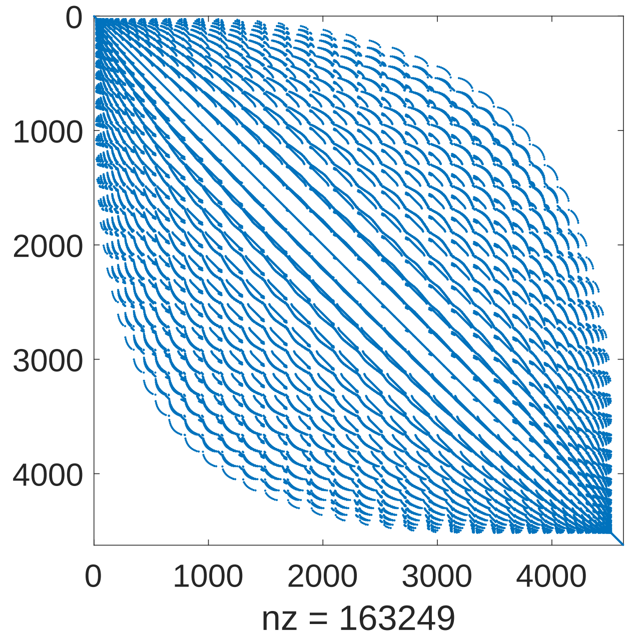


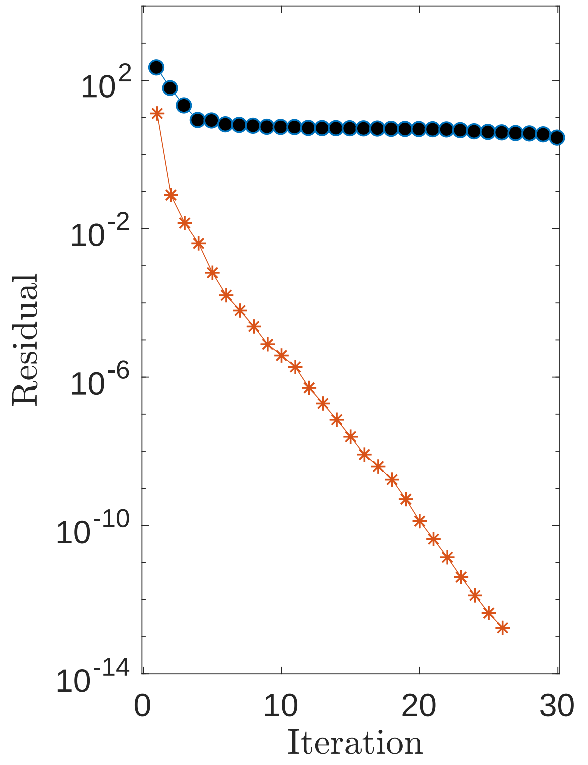
Space-Time PS Collocation Method: 2D+t, linear case



$$
\begin{aligned}
PDE: &\quad u_t = \Delta u + F(x,y,t)\\
&\quad (x,y,t) \in \Omega \times (0,T]\\
IC: &\quad u(x,y,0) = f(x,y)\\
BC: &\quad u(\partial\Omega,t) = g(\partial\Omega,t)
\end{aligned}
$$

kron's disease is worse in $2D + t$ case.





```
P = symrcm(PLinop);

L = gpuArray(Linop(P,P));

PL = gpuArray(PLinop(P,P));

r = gpuArray(rhs(P));

MAXITER = 30; TOL = 1e-14; RESTART = [];

[Ugpu,FLAG,RELRES,ITER,RESVEC] = ...

gmres(L,r,RESTART,TOL,MAXITER,PL);

U(P) = gather(Ugpu);
```

Space-Time PS Collocation Method: 1D+t, nonlinear case

Human tear film dynamics: 1D model: see H. et. al 2007

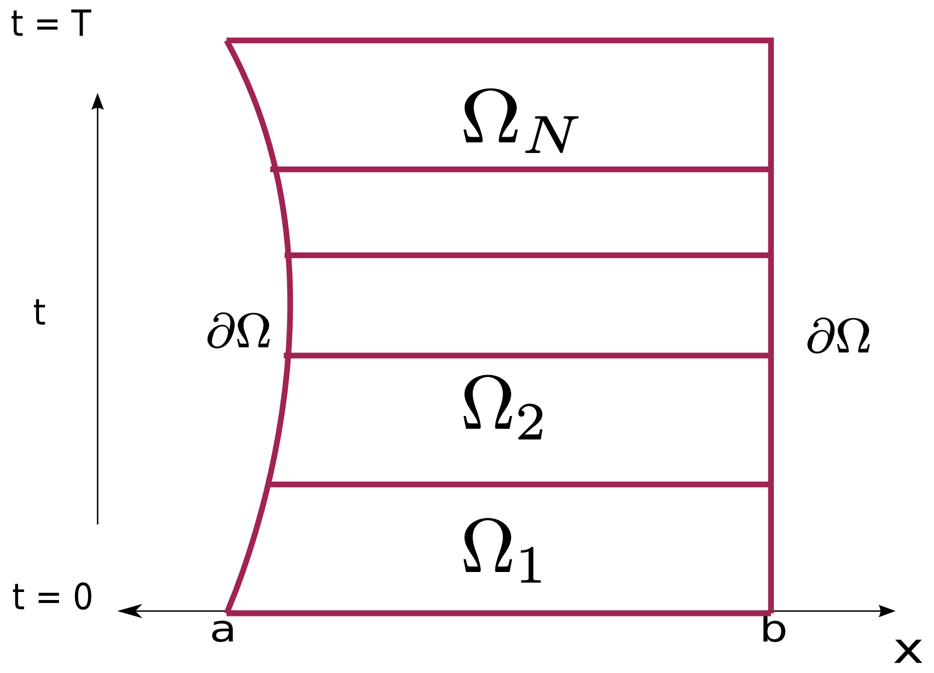
$$h_t + q_x = 0 \text{ on } X(t) \le x \le 1,$$

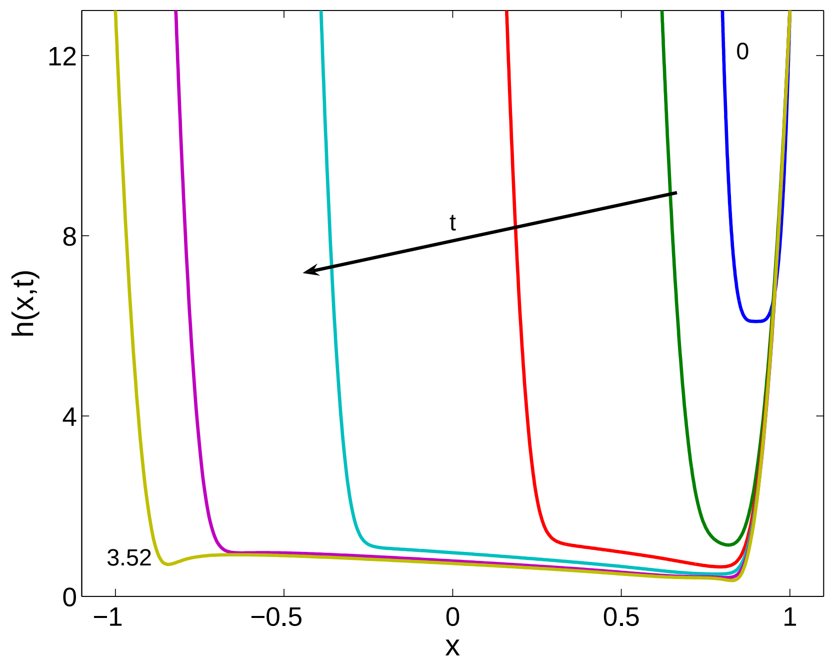
where

$$q(x,t) = Sh_{xxx}\left(\frac{h^3}{3} + \beta h^2\right)$$

Boundary conditions

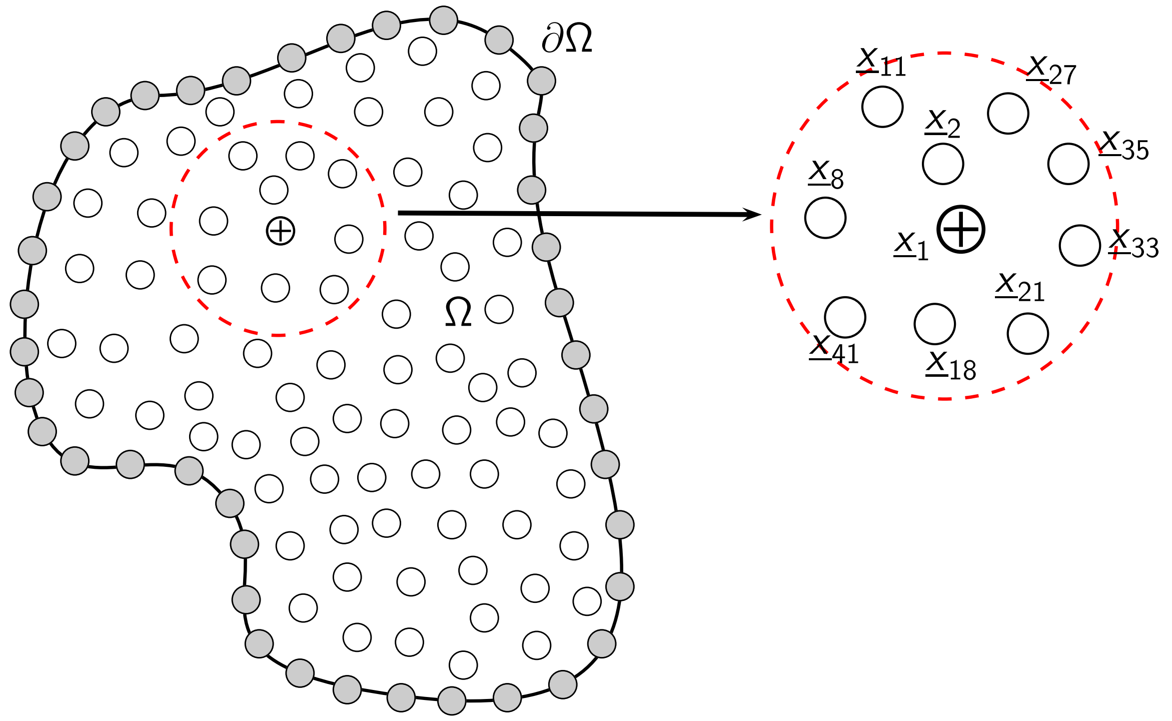
$$h(X(t),t) = h(1,t) = h_0 \quad q(X(t),t) = X_t h_0 + Q_{top} \quad q(1,t) = -Q_{bot}.$$





Advance the solution in space-time domain: Slab by Slab (Show MATLAB).

RBF-FD Differentiation Matrices



$$s_j(\underline{x}) = \sum_{k=1}^{n_{\text{loc}}} \lambda_k \phi^k(\underline{x}),$$

where $\phi^k(\underline{x})$ is a radial basis function centered at $\underline{x}_k$.
Or in Lagrange formulation as

$$s_j(\underline{x}) = \sum_{k=1}^{n_{\text{loc}}} \Psi^k(\underline{x}) u_k,$$

where

$$\underline{\Psi} = \begin{bmatrix}\Psi^1(\underline{x}) & \cdots & \Psi^{n_{\text{loc}}}(\underline{x})\end{bmatrix} = \begin{bmatrix}\phi^1(\underline{x}) & \cdots & \phi^{n_{\text{loc}}}(\underline{x})\end{bmatrix}\begin{bmatrix}A^{-1}\end{bmatrix},$$

$$\underline{\Psi}_x = \begin{bmatrix}\Psi^1_x(\underline{x}) & \cdots & \Psi^{n_{\text{loc}}}_x(\underline{x})\end{bmatrix} = \begin{bmatrix}\phi^1_x(\underline{x}) & \cdots & \phi^{n_{\text{loc}}}_x(\underline{x})\end{bmatrix}\begin{bmatrix}\mathrm{A}^{-1}\end{bmatrix},$$

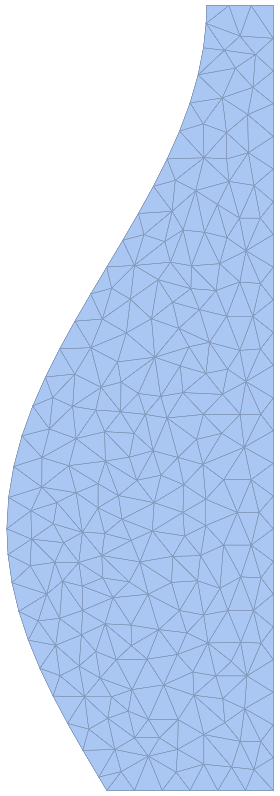
The matrix A with entries

$$a_{\ell k} = \phi^k(\underline{x}_\ell), \qquad \ell, k = 1, \ldots, n_{\text{loc}}$$

is *local RBF interpolation matrix*.
BYODM: Bring Your Own Differentiation Matrices

Getting the space-time domain

This is probably for programming on a lazy Sunday: Use MATHEMATICA's **DiscretizeRegion** family commands. Surprisingly, MATHEMATICA has many built-in funky domains too. This is also useful if you want to compare results with finite-element.



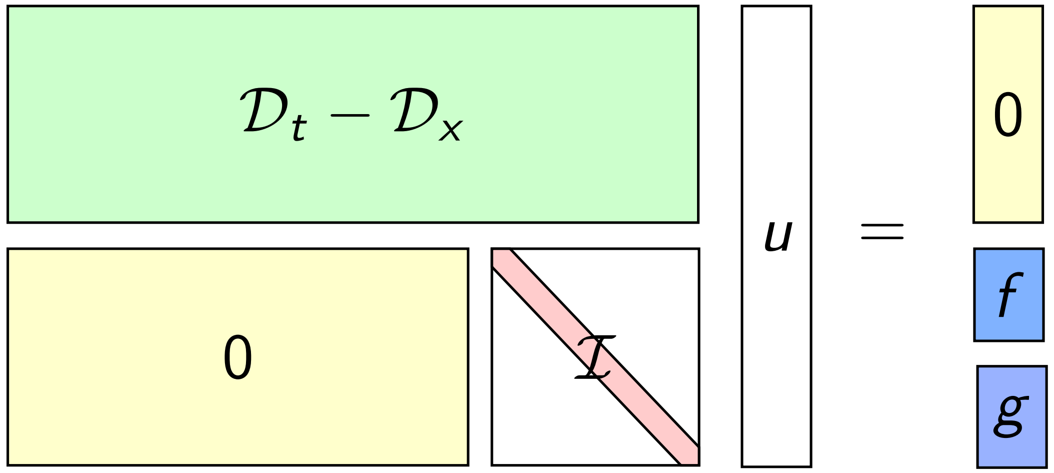
```
R = ImplicitRegion[-0.6 Sin[t] <= x, {{x, -1, 1},
       {t, 0, 1.5 Pi}}];
ev = DiscretizeRegion[R];
pts = MeshCoordinates[ev];
Export["spacetimedom.mat", pts];
```

To obtained boundary points, you can use MATHEMATICA or **boundary** command in MATLAB.

t+1D Advection Example

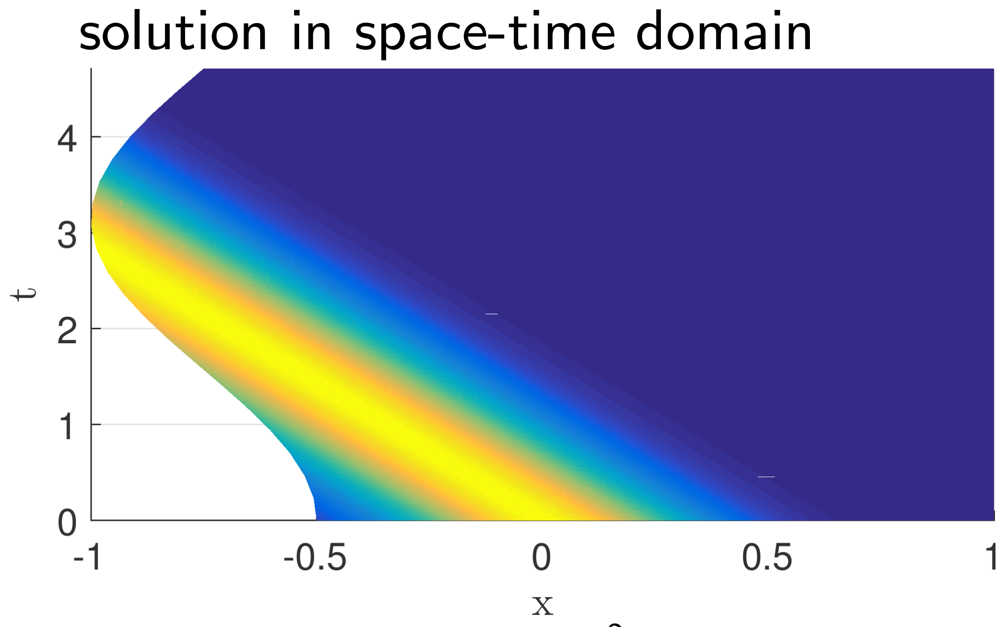
$$
\begin{aligned}
PDE: \quad & u_t = u_x \\
& (x,t) \in [X(t),1) \times (0,T] \\
IC: \quad & u(x,0) = f(x) \\
BC: \quad & u(1,t) = g(t)
\end{aligned}
$$

IMQ-RBF: $\frac{1}{\sqrt{1+(\varepsilon r)^2}}$. $r^2 = (x-x_i)^2 + (t-t_i)^2$

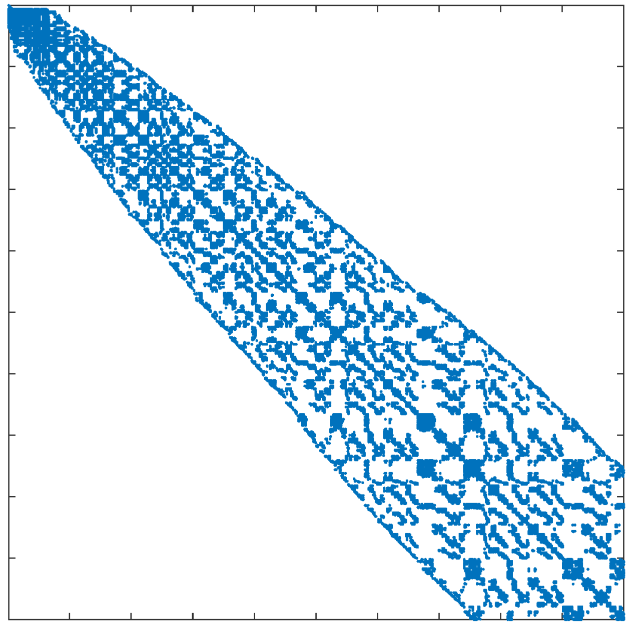


```
P = symrcm(L); u(P) = L(P,P)\RHS(P);
```
or

```
MAXITER = 20; TOL = 1e-13; RESTART = [];
[ML,MU] = ilu(L(P,P),struct('type','ilutp','droptol',1e-6));
u(P) = gmres(L(P,P),RHS(P),RESTART,TOL,MAXITER,ML,MU);
```



$f(x) = e^{-10(x-0.15+0.35y)^2}$, $g(t) = 0$



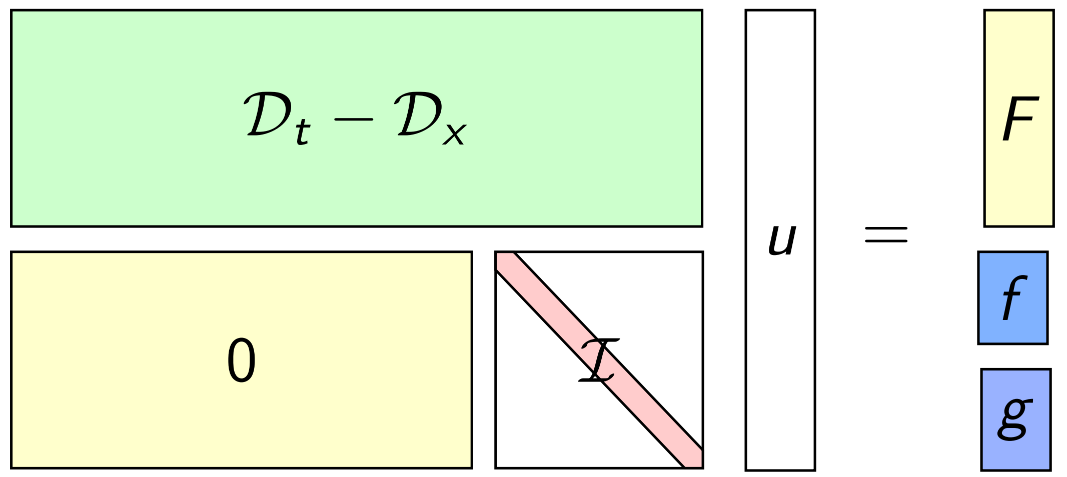
portion of system matrix
after applying MATLAB symrcm

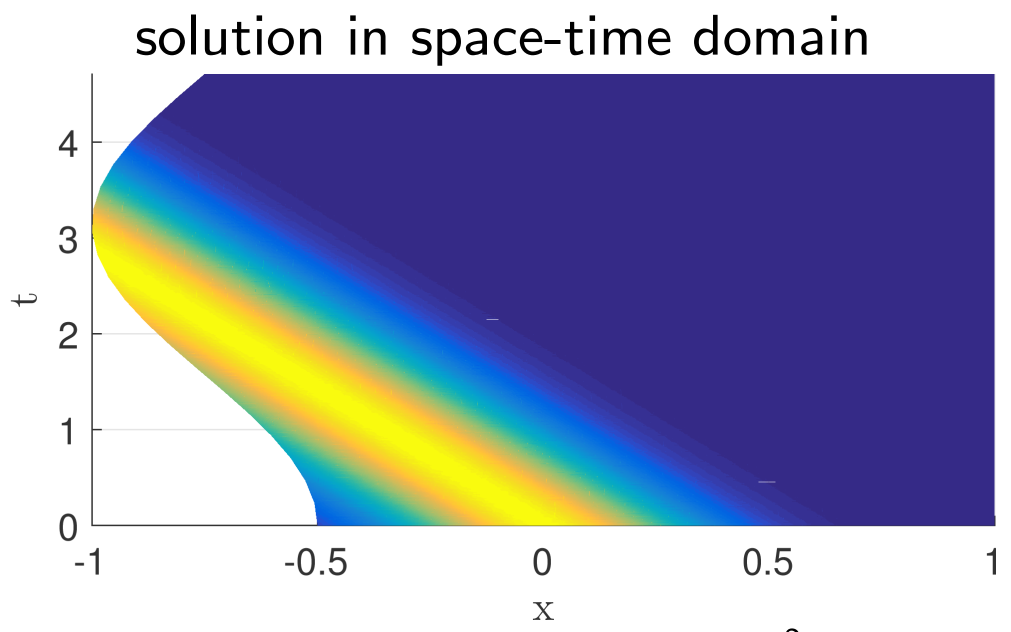
t+1D Advection Example

$$\begin{aligned} PDE: &\quad u_t = u_x + F(x,t) \\ &\quad (x,t) \in [X(t),1) \times (0,T] \\ IC: &\quad u(x,0) = f(x) \\ BC: &\quad u(1,t) = g(t) \end{aligned}$$

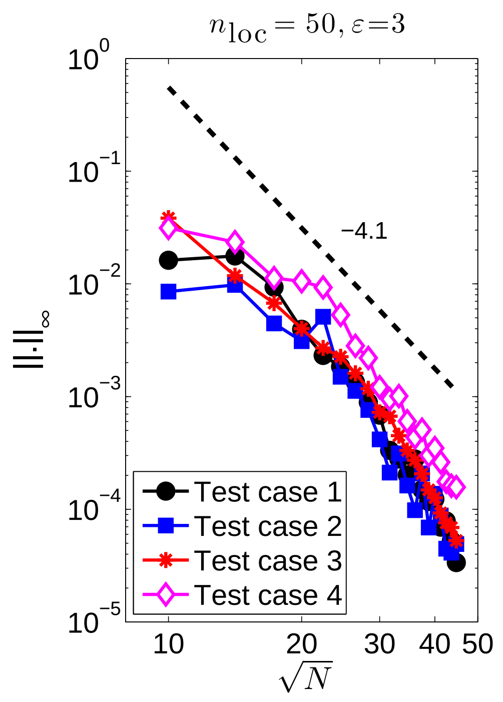
IMQ-RBF: $\frac{1}{\sqrt{1+(\varepsilon r)^2}}$. $r^2 = (x-x_i)^2 + (t-t_i)^2$

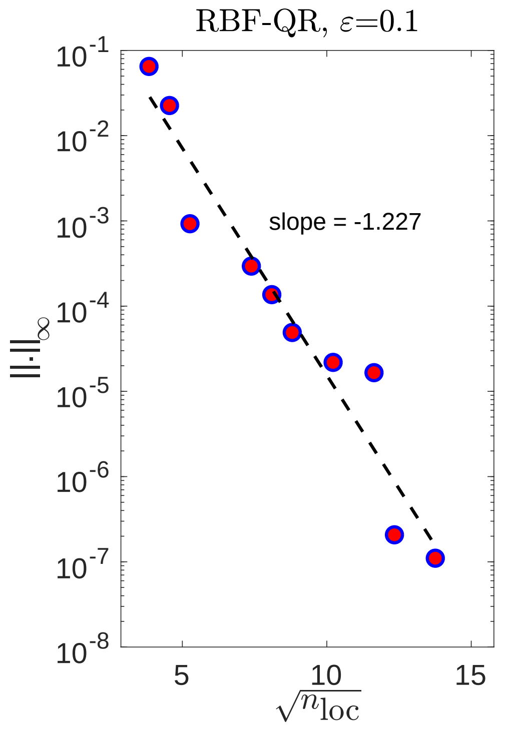
Get RBF-QR diffmat from Elisabeth's website.





$$f(x) = e^{-10(x-0.15+0.35y)^2}$$

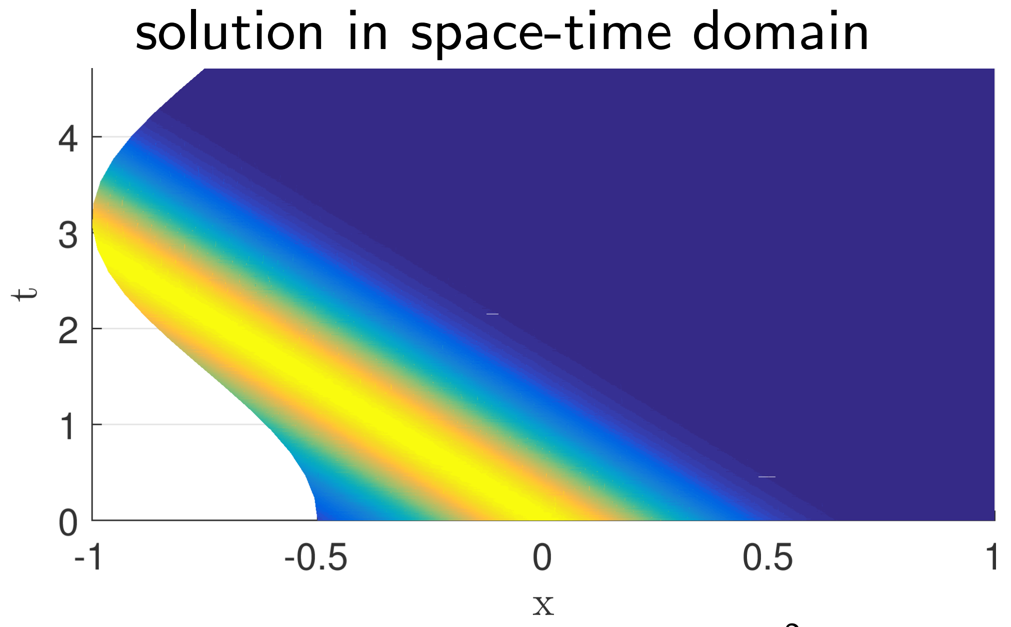




t+1D Advection with Variable Speed Example

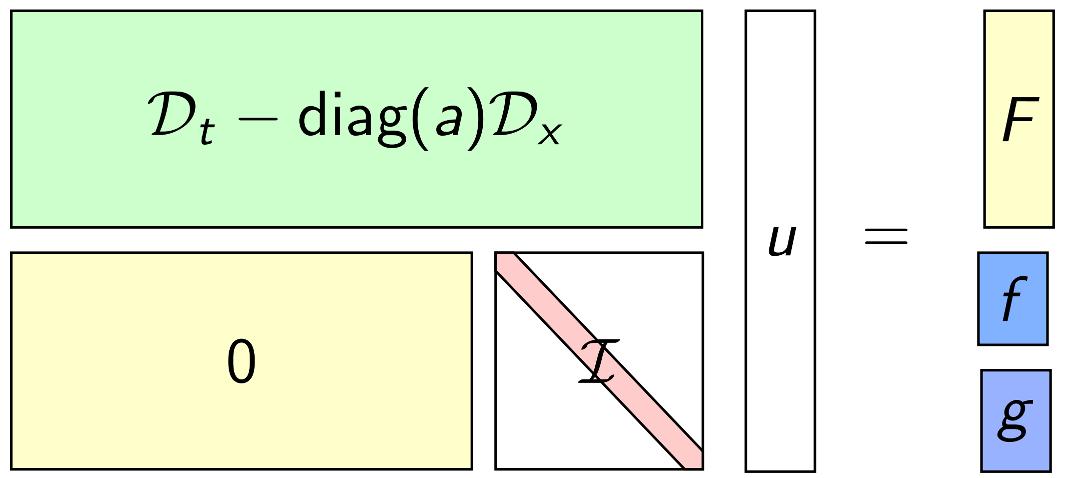
$$
\begin{aligned}
PDE: &\quad u_t = a(x,t)u_x + F(x,t)\\
&\quad (x,t) \in [X(t),1) \times (0,T]\\
IC: &\quad u(x,0) = f(x)\\
BC: &\quad u(1,t) = g(t)
\end{aligned}
$$

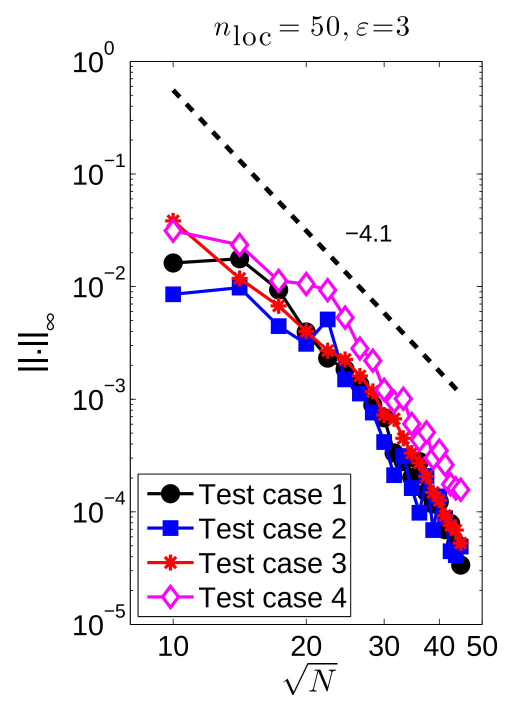
$$a(x,t) = \exp((1+t)(1+\cos(3x))$$

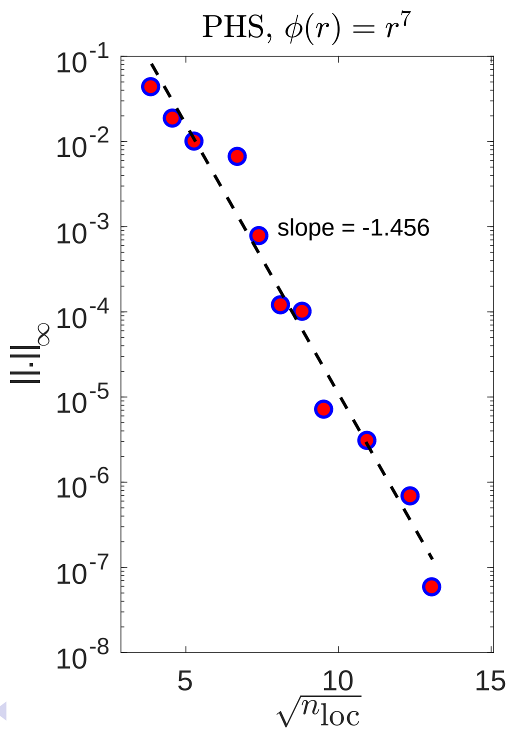


$$f(x) = e^{-10(x-0.15+0.35y)^2}$$

Bribe Varun for PHS diffmat.

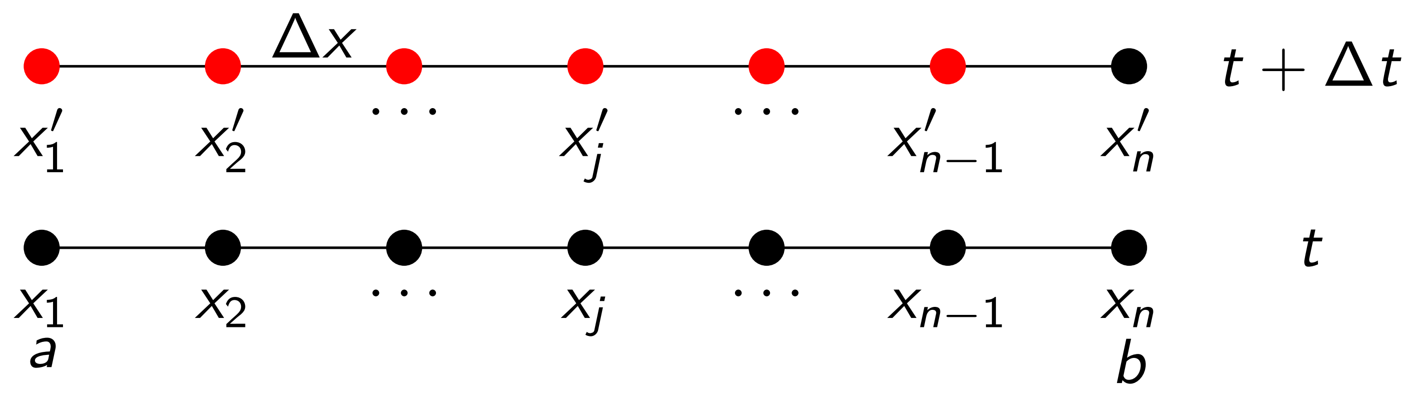






Analyzing Stability ?

Let's take a look at one step (2 levels) space-time global RBF method.

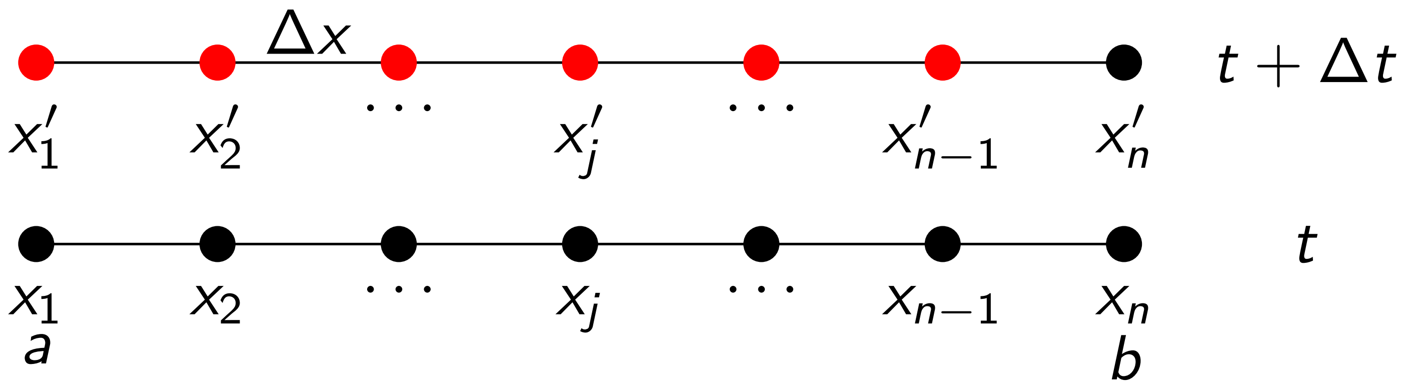


for a simple 1-D advection equation

$$\begin{aligned}\text{PDE:} \quad & \frac{\partial u}{\partial t} = \frac{\partial u}{\partial x} \quad \text{for } x \in [a, b) \\ \text{IC:} \quad & u(x,0) = u_0(x) \quad \text{when } t = 0 \\ \text{BC:} \quad & u(b,t) = g(t) \quad \text{at } x = b\end{aligned}$$

$$u(x) = \sum_{j=1}^{n} \lambda_j \phi(\varepsilon \|x - x_j\|) + \sum_{j=1}^{n} \lambda_j' \phi(\varepsilon \|x - x_j'\|),$$

where $\{x_j\}$ and $\{x_j\}$ are centers at the old time level and new time level respectively.



Our goal is to find the unknowns $\{\lambda_j\}$ and $\{\lambda_j'\}$. This can be done by enforcing initial and boundary data and satisfying the PDE at the interior points that lead to solving system of linear equations

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]\left[\begin{array}{c} \lambda_1 \\ \vdots \\ \lambda_n \\ \hline \lambda_1' \\ \vdots \\ \lambda_n' \end{array}\right] = \left[\begin{array}{c} u_o(x_1) \\ \vdots \\ u_o(x_n) \\ \hline 0 \\ \vdots \\ g(t) \end{array}\right]$$

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]\left[\begin{array}{c} \lambda_1 \\ \vdots \\ \lambda_n \\ \hline \lambda'_1 \\ \vdots \\ \lambda'_n \end{array}\right] = \left[\begin{array}{c} u_o(x_1) \\ \vdots \\ u_o(x_n) \\ \hline 0 \\ \vdots \\ g(t) \end{array}\right]$$

The block matrices A, B, C, D are all $n \times n$ matrices with elements:

- $A_{ij} = \phi(\varepsilon\|x_i - x_j\|)$
- $B_{ij} = \phi(\varepsilon\|x_i - x'_j\|)$
- $C_{ij} = \mathcal{L}\phi(\varepsilon\|x'_i - x_j\|)$
- $D_{ij} = \mathcal{L}\phi(\varepsilon\|x'_i - x'_j\|)$

for all $i, j = 1, \cdots, n$ and $\mathcal{L} := \frac{\partial}{\partial t} - \frac{\partial}{\partial x}$. The last row C and D must be slightly modified to satisfy the boundary condition at $x'_n = b$.

Amplification Matrix and Stability Region

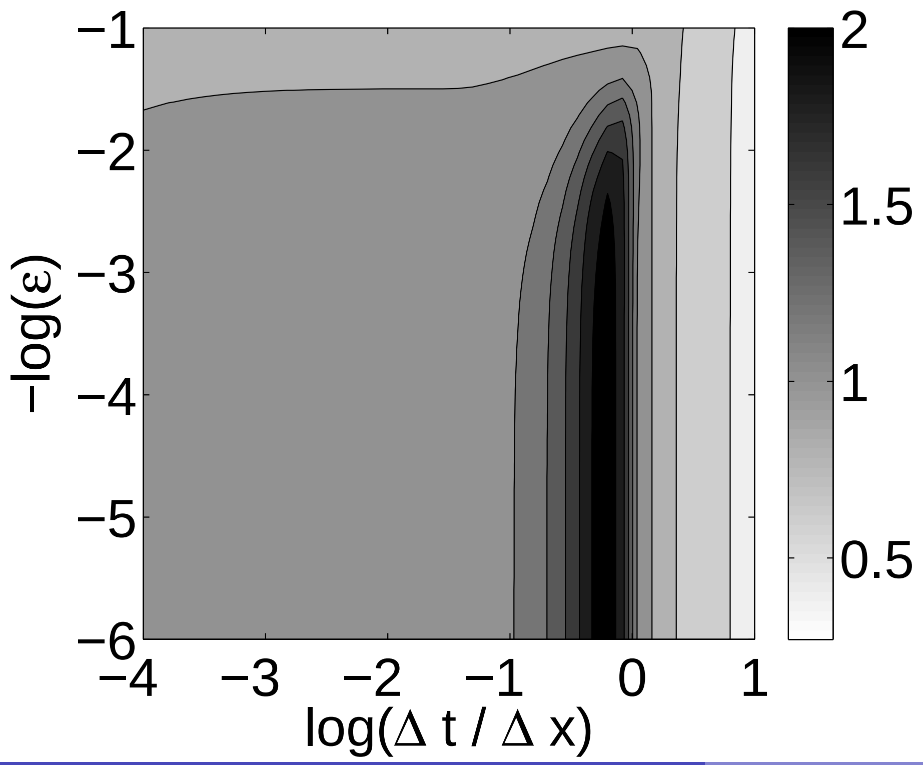
The process of marching in time to the new time level is given by

$$\begin{bmatrix} u(x_1') \\ \vdots \\ u(x_n') \end{bmatrix} = \begin{bmatrix} & & \\ & G & \\ & & \end{bmatrix} \begin{bmatrix} u(x_1) \\ \vdots \\ u(x_n) \end{bmatrix}$$

where

$$G = \left[\; B \mid A \;\right] \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]^{-1} \left[\begin{array}{c} I \\ \hline 0 \end{array}\right],$$

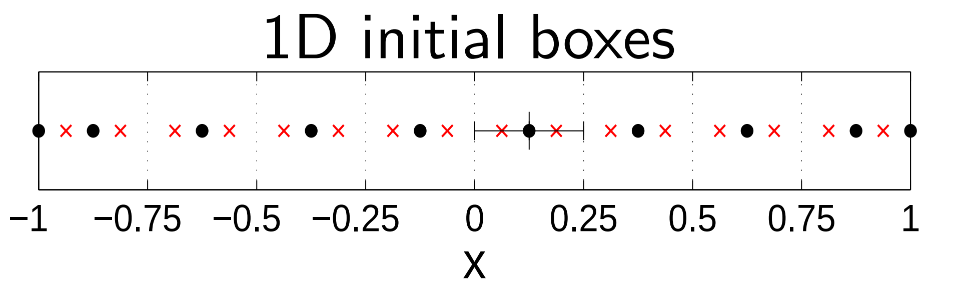
and I is an $n \times n$ identity matrix. The method is numerically stable if spectral radius $\rho(G) < 1$.

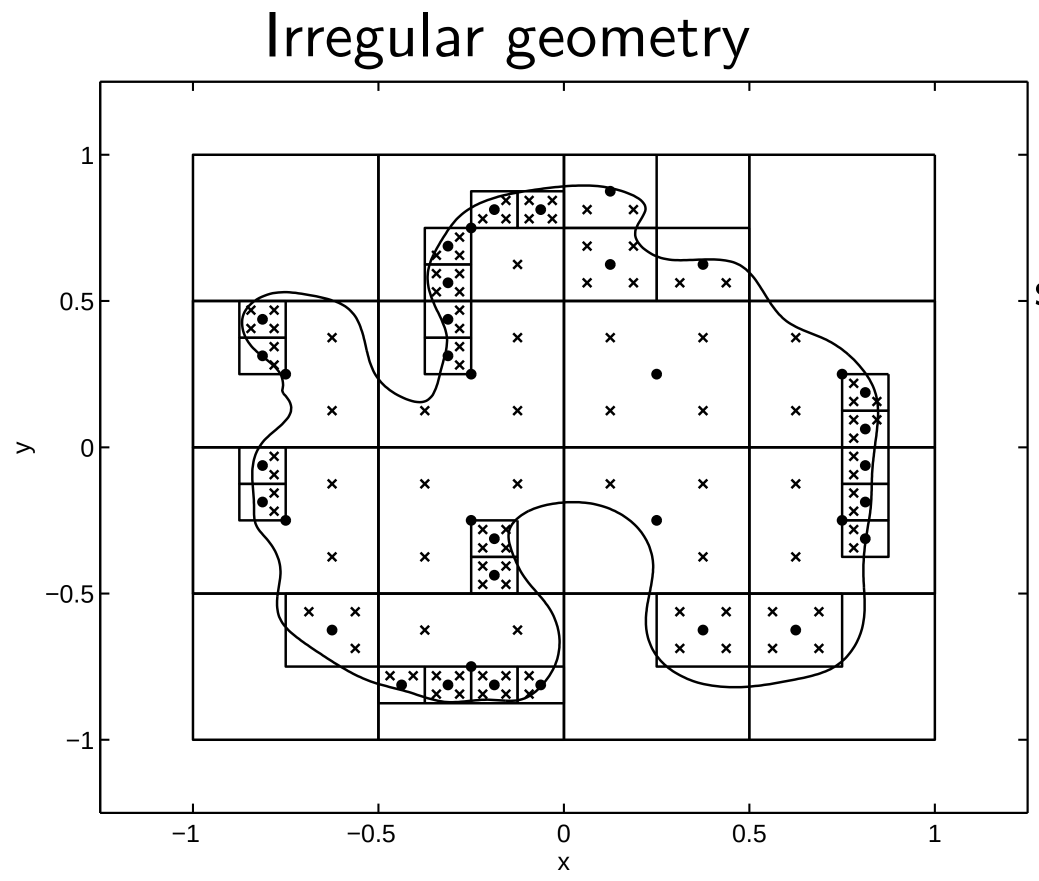


IMQ, $g(t) = 0$, $N = 50$: to avoid blowing up the solution, the ratio of $\Delta t/\Delta x$ vs shape parameter ε must be away from the darker alley in the the stability region, i.e we want to avoid $\rho(G) \geq 1$

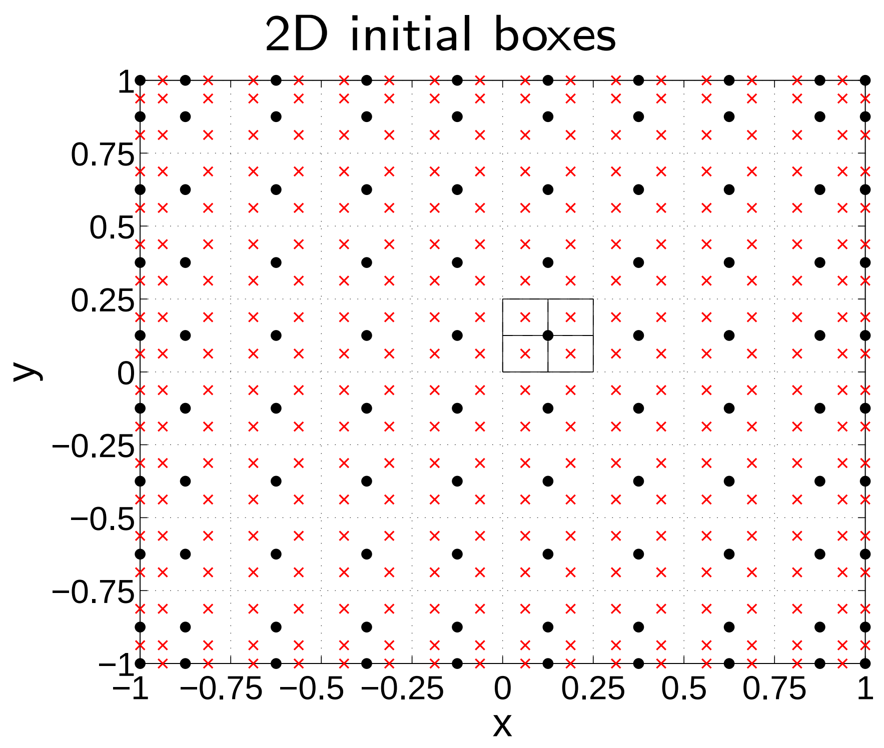
Adaptivity for BVP based on Residual subsampling

1. Initial coarse collection of nonoverlapping regular boxes in R^d that cover the domain Ω of interest.
2. Geometric adaptation.
3. Refining and Coarsening steps.



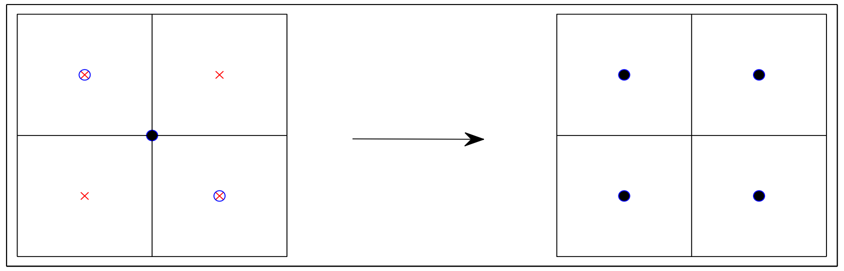


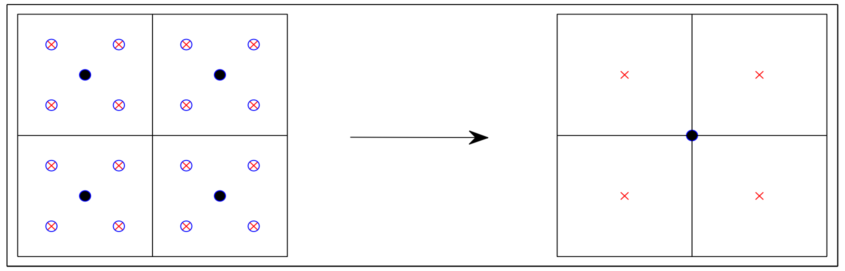
see



Driscoll & H (2007)

Rules of refining and coarsening centers

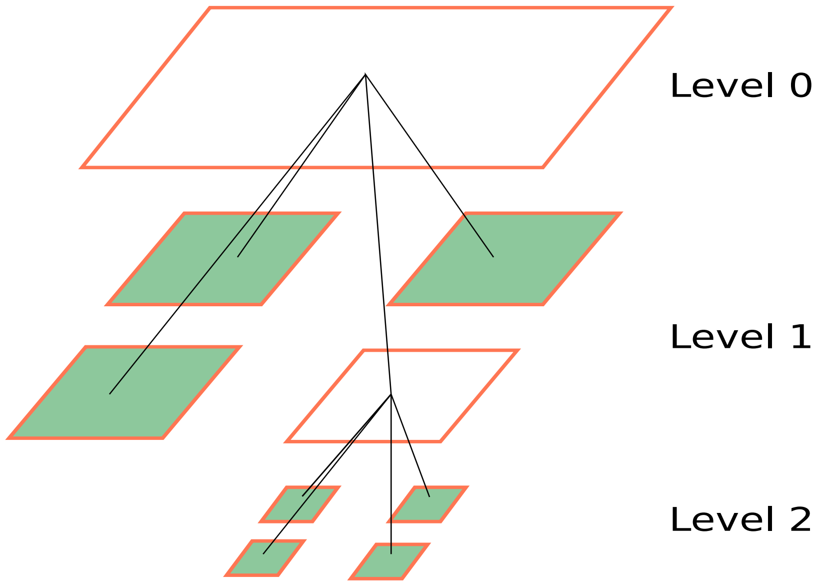




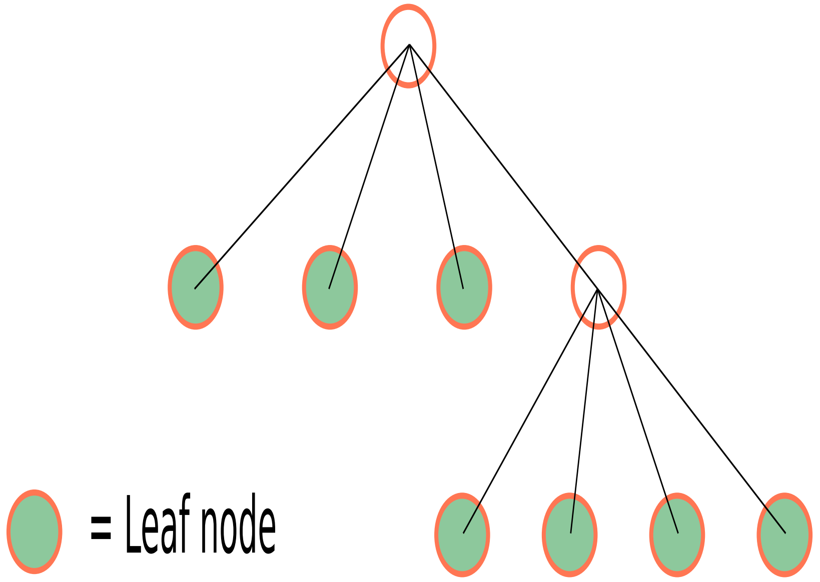
Refinement strategy: converting all check points if any of them have residual errors are greater than θ_r described as $\otimes$ into RBF centers as dots and remove its parent.

Coarsening strategy: reactivate all RBF centers if all of its grand children have residual errors less than θ_c described as $\otimes$.

a layout of quadtree.

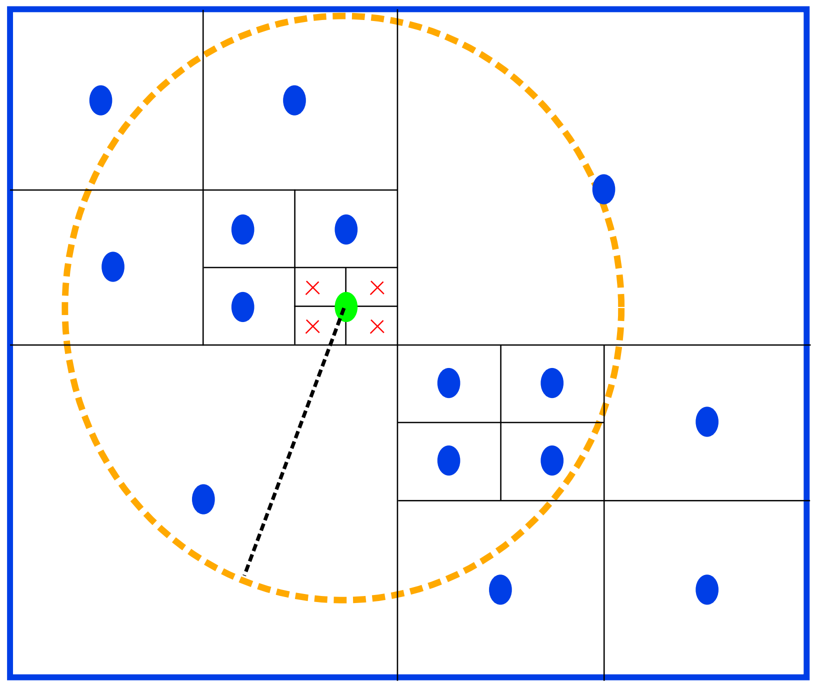


With this rule, centers are located as leaves.



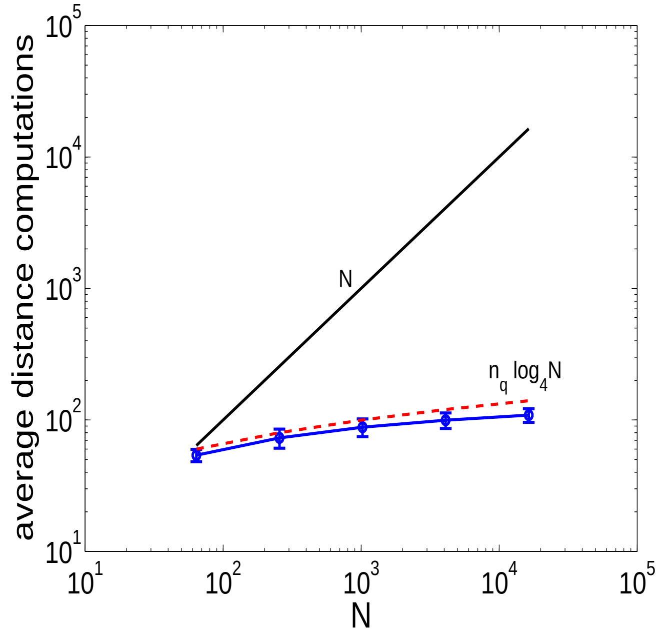
Depth first search algorithm

- Pruning device to save computing pairwise distances.:$\mathcal{O}(n_q \log(N))$ instead of $\mathcal{O}(N)$ per query point.
- Partial updates for lists of neighbors.
- Embarassingly parallel neighbors' search.

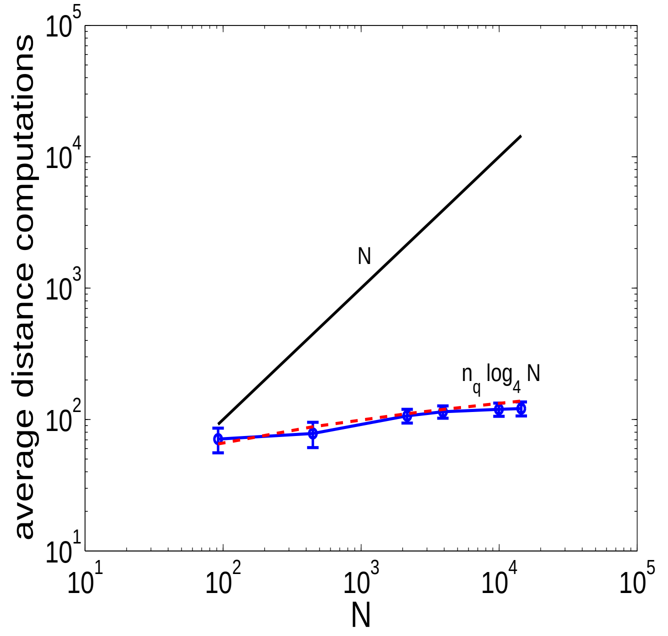


Values at $\times$ are computed using local RBF interpolant of the box whose midpoint is the parent node of the check points.

Uniform nodes distribution



Some non-uniform nodes distribution

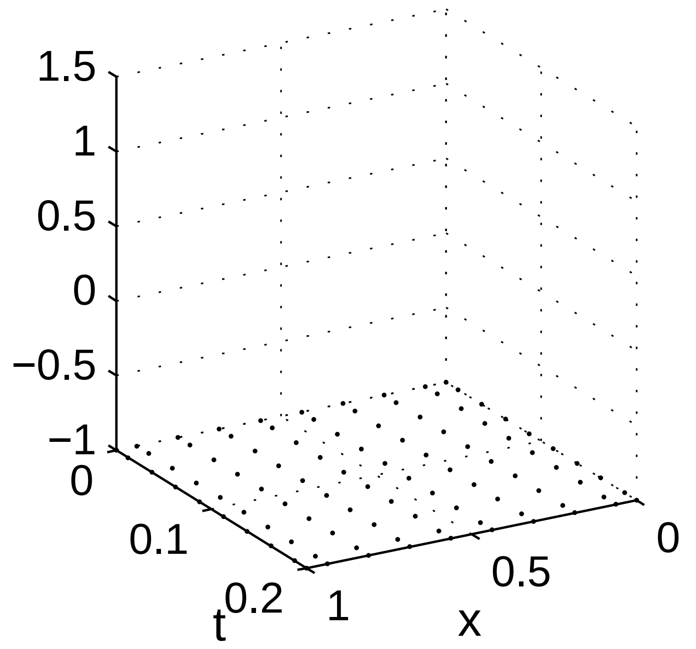


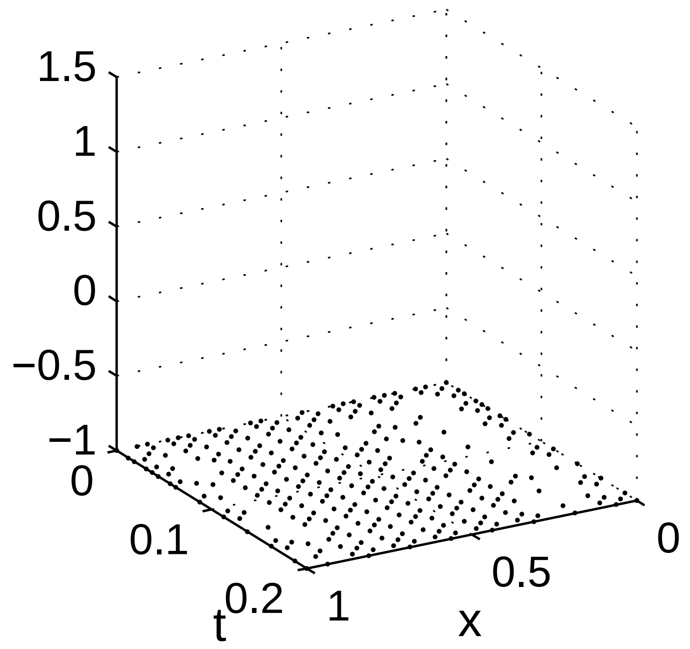
t+1D Nonlinear Example

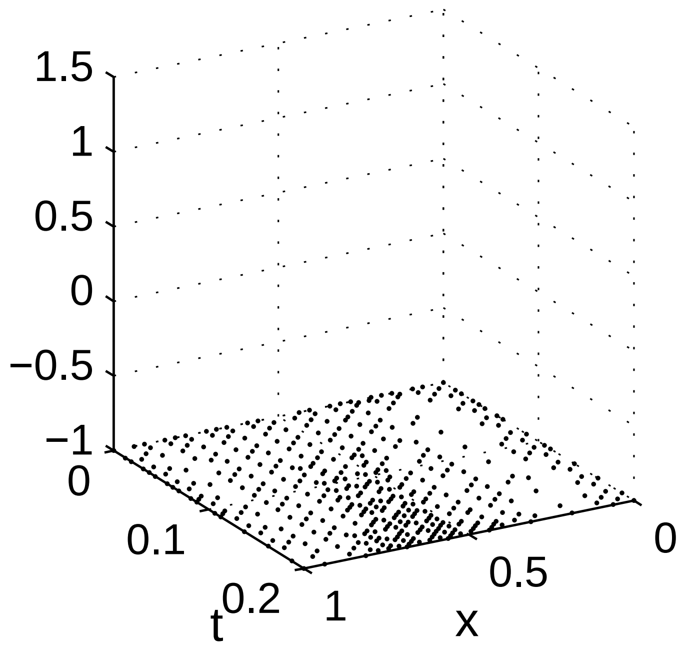
Burgers' Equation

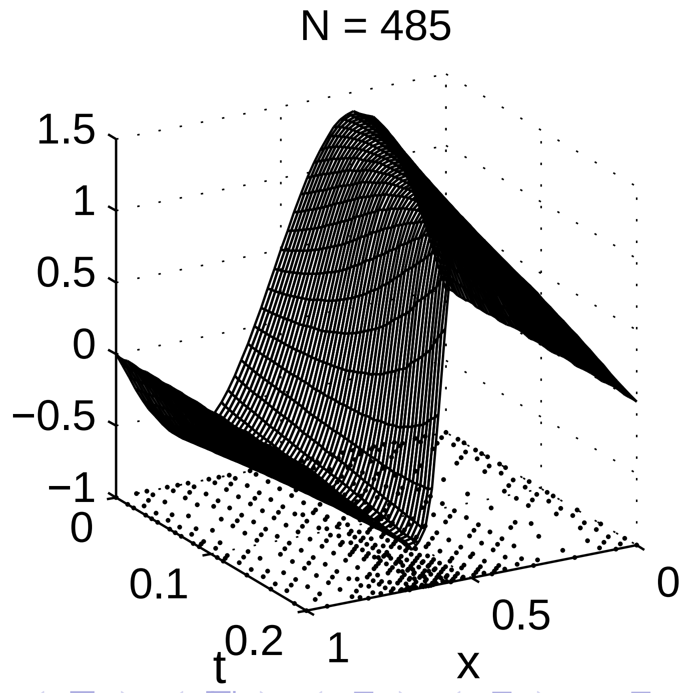
$$\upsilon u_{xx} - u u_x = u_t, \qquad 0 < x < 1$$
$$u(0,t) = u(1,t) = 0$$
$$u(x,0) = \sin(2\pi x) + \tfrac{1}{2}\sin(\pi x).$$
$$\text{where, } \upsilon = 10^{-3}$$

MATLAB's **fsolve** is used to solve the nonlinear system. Jacobian file is provided.









Dealing with Multiple Boundary Conditions

PDE : Tear film PDE in terms of h

$(x,t) \in [X(t),1) \times (0,T]$

IC : $h(x,0) = f(x)$

BC : $h(1,t) = h(X(t),t) = h_0$

$h_{xxx}(1,t) = g_1(t)$

$h_{xxx}(X(t),t) = g_2(X(t),t)$

PDE : $h_t = Sq_x$,S is a constant

$q =$ nonlinear flux

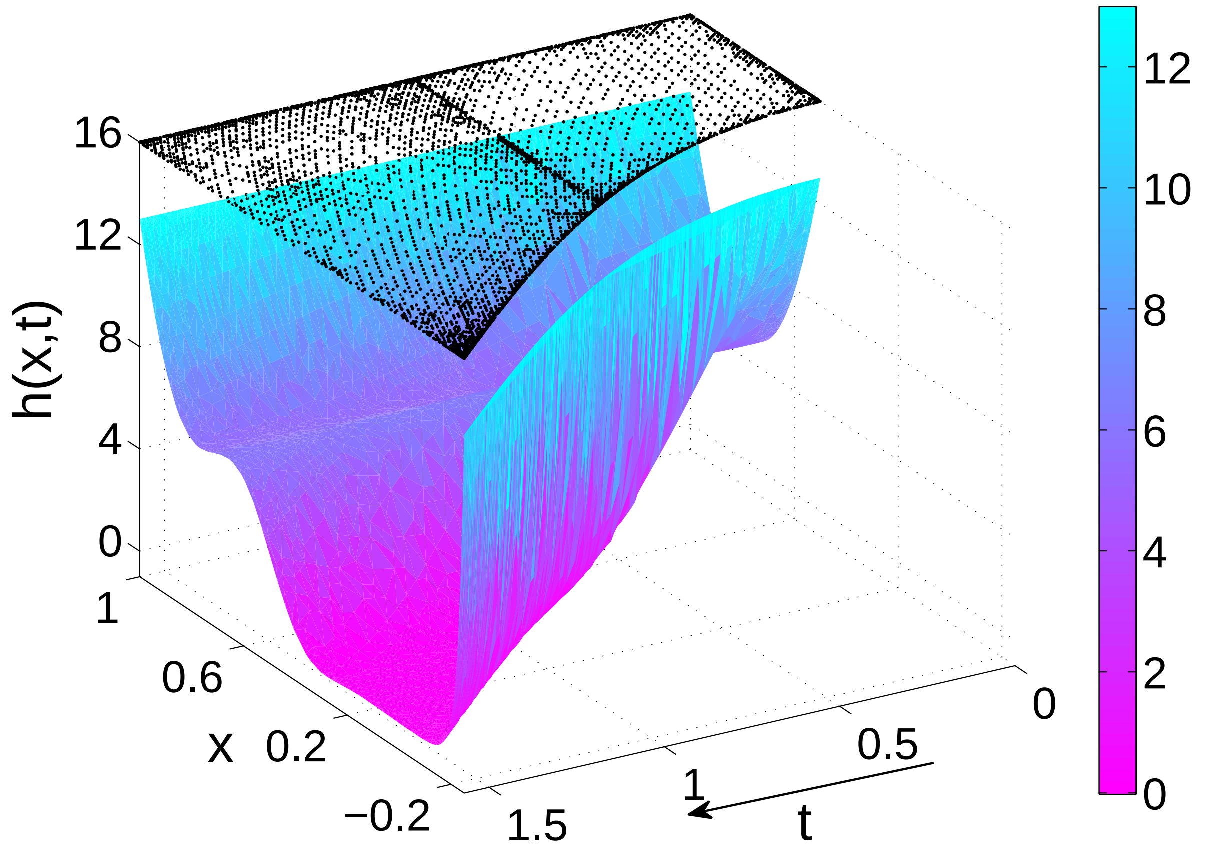
$(x,t) \in [X(t),1) \times (0,T]$

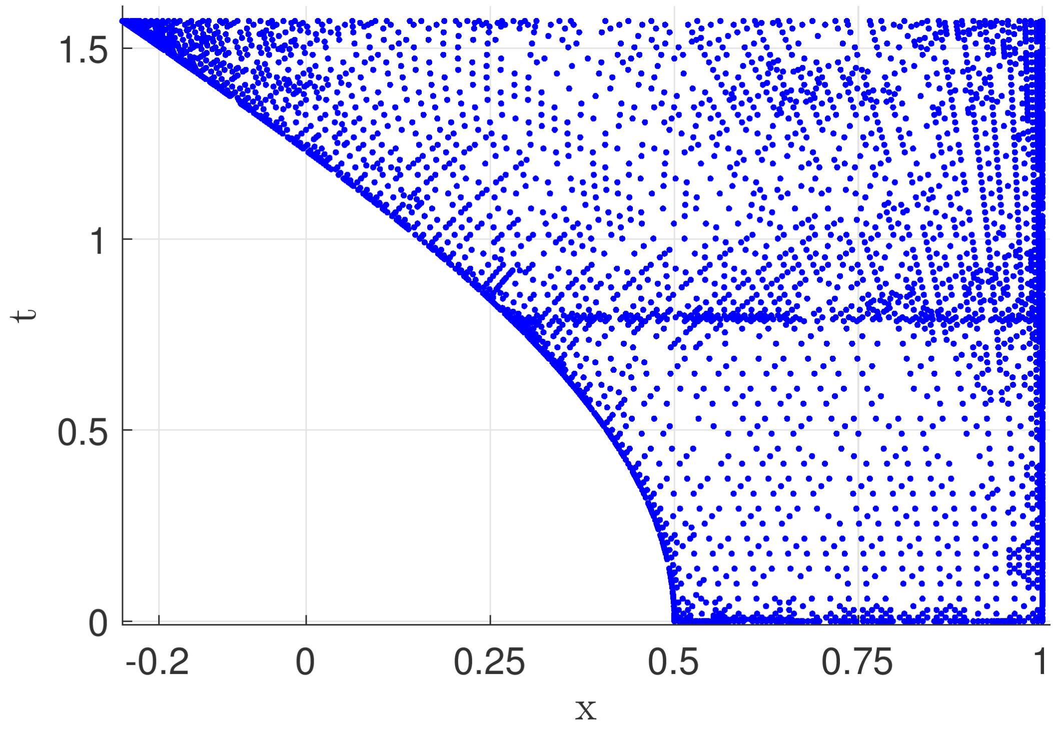
IC : $h(x,0) = f(x)$

BC : $h(1,t) = h(X(t),t) = h_0$

$q(1,t) = g_1(t)$

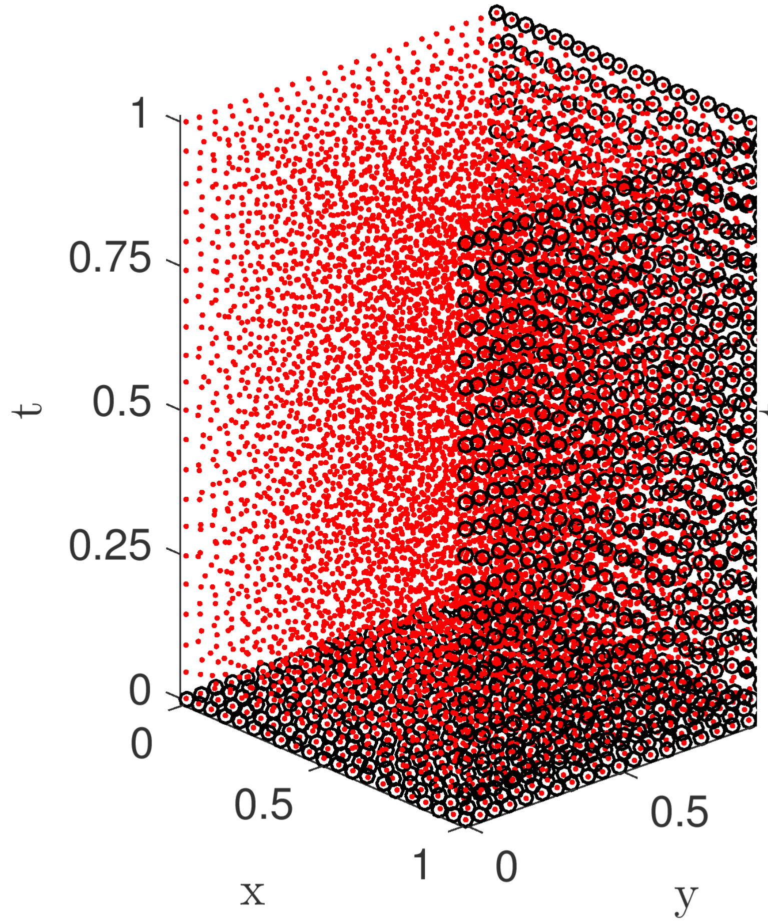
$q(X(t),t) = g_2(X(t),t)$

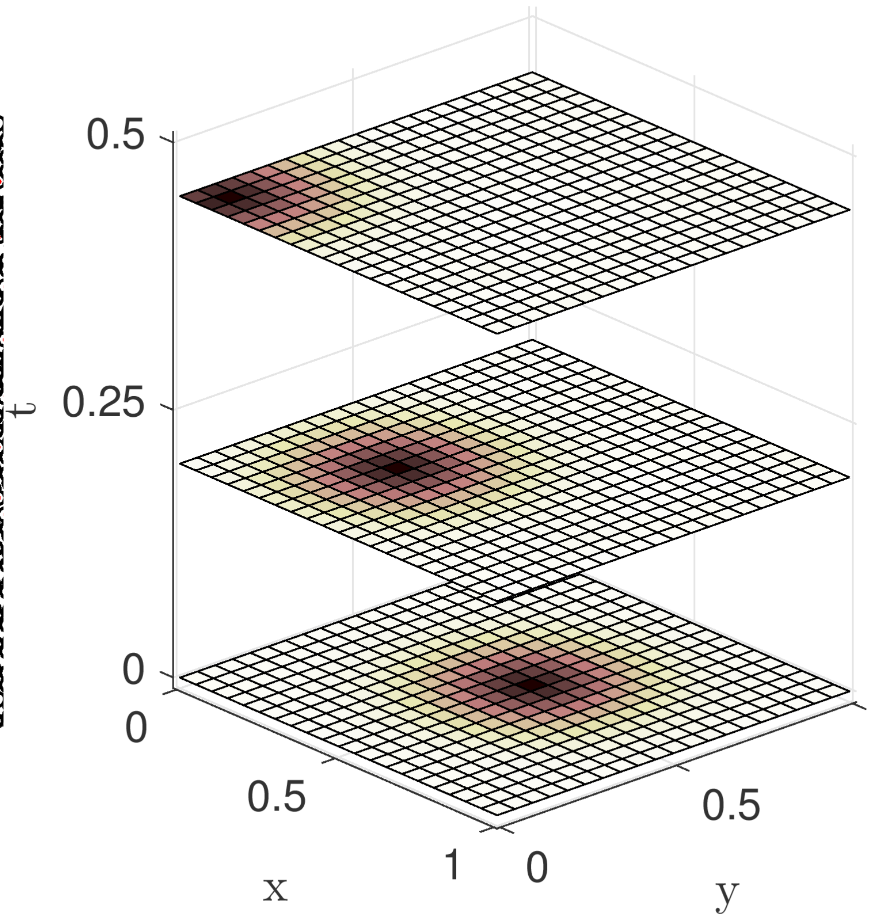


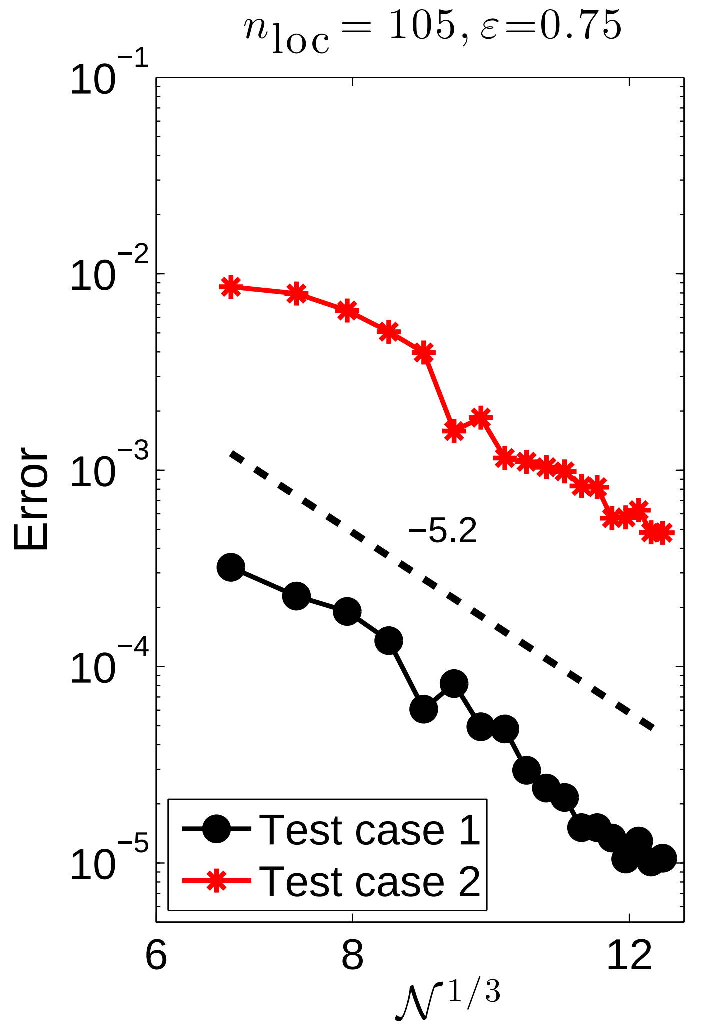


t+2D Advection Example

$$u_t = 0.5u_x + 0.75u_y + F(x,y,t) \quad (x,y) \in [0,1) \times [0,1)$$
$$u(1,y,t) = f_1(1,y,t) \quad u(x,1,t) = f_2(x,1,t)$$
$$u(x,y,0) = g(x,y)$$







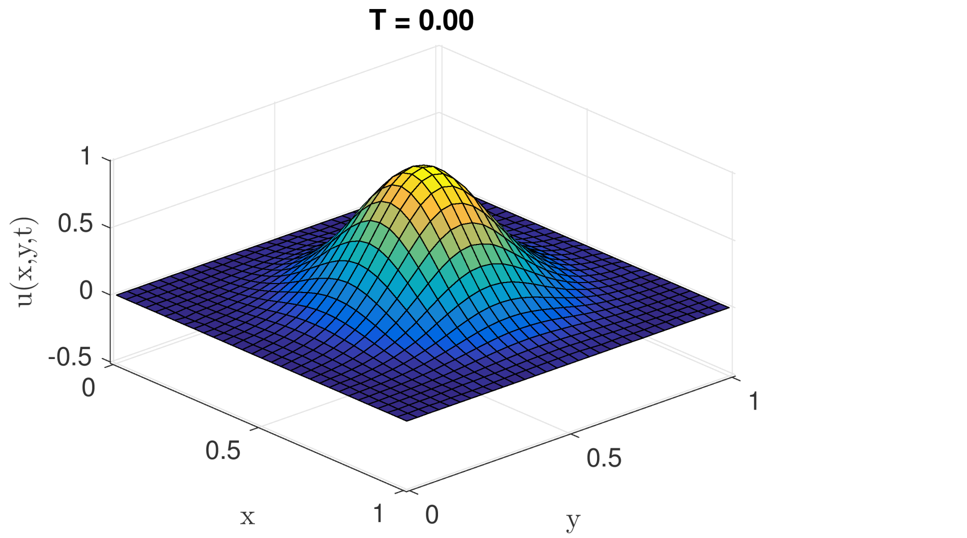
t+2D Wave Example

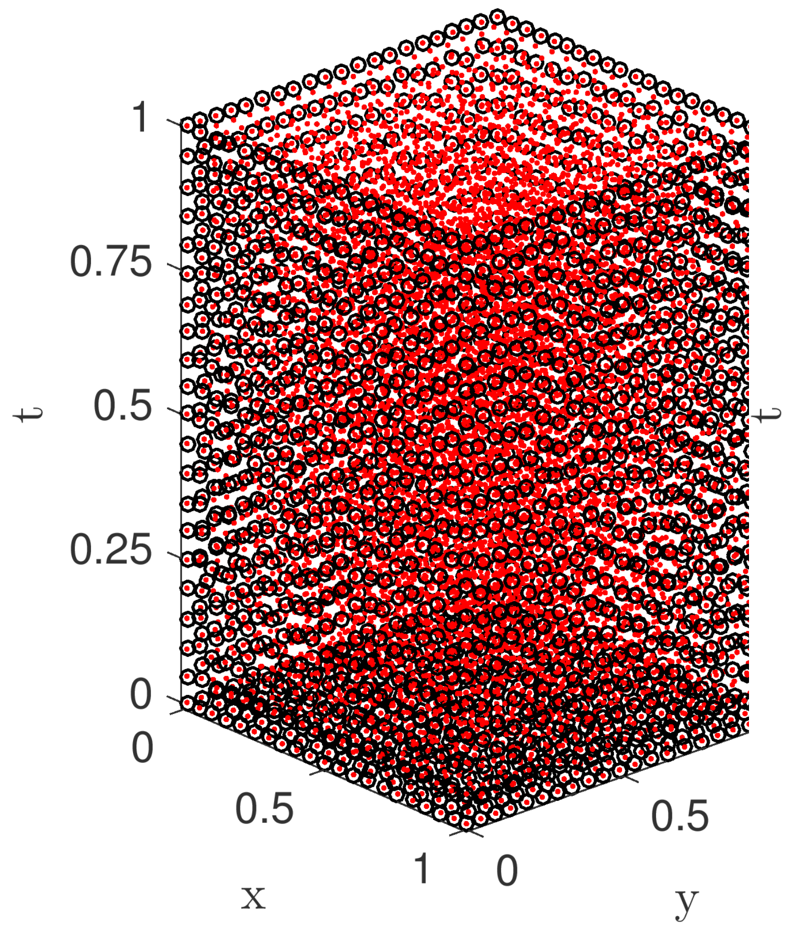
$$u_{tt} = \Delta u \quad (x, y) \in (0,1) \times (0,1)$$

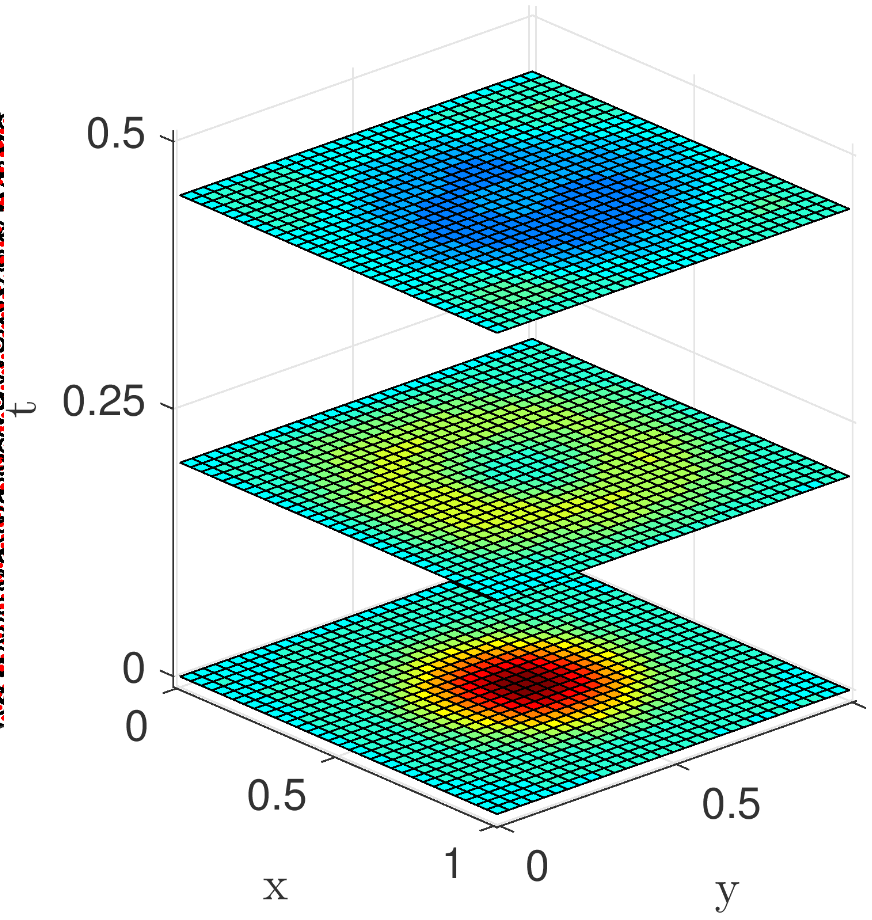
$$u(x,y,t) = 0 \quad \text{at the boundary}$$

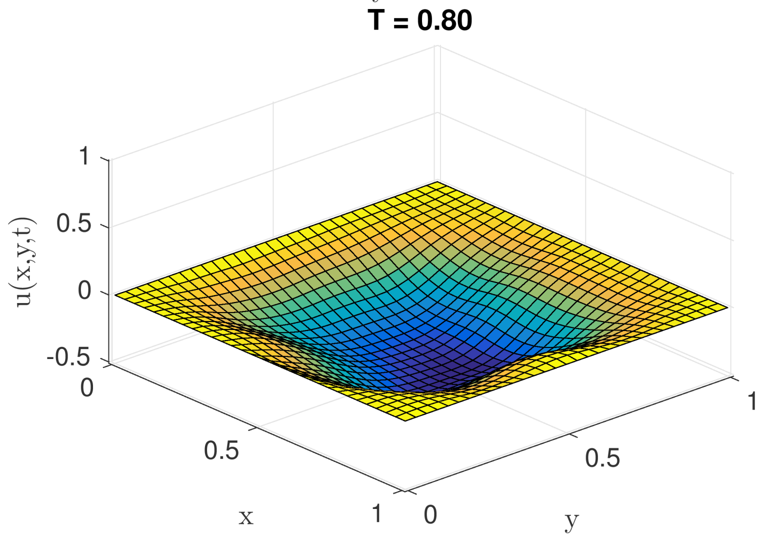
$$u(x,y,0) = g(x,y)$$

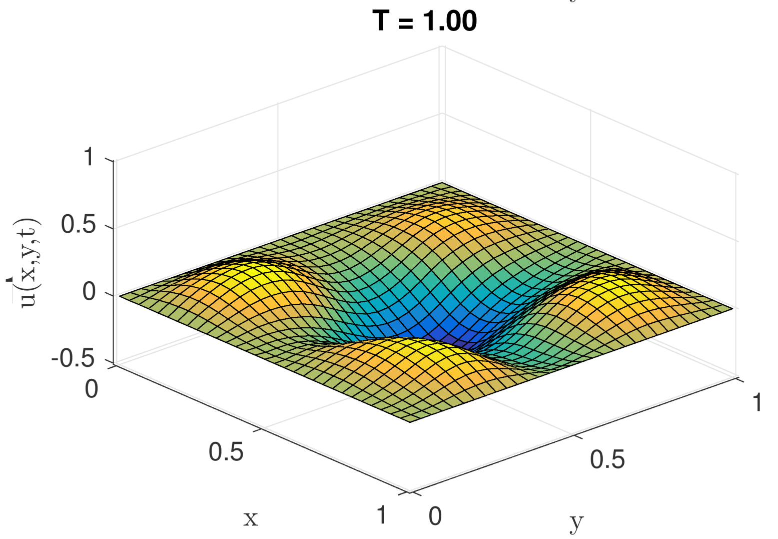
$$u_t(x,y,0) = 0$$







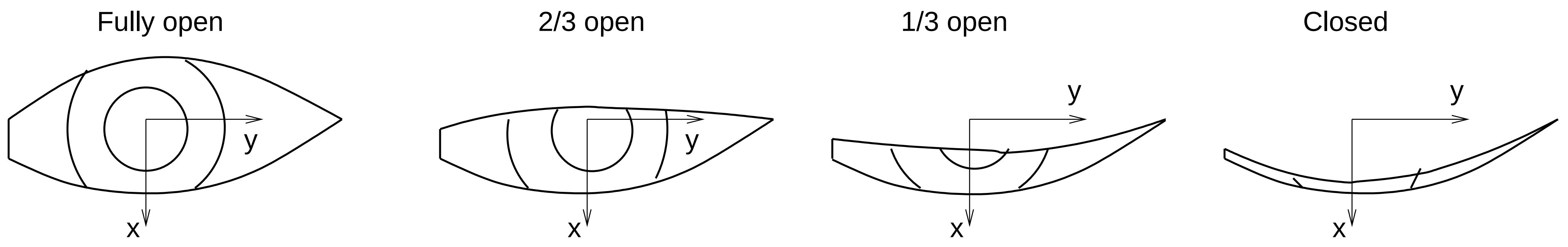




extra ghost/fictitious points for enforcing u_t

On-going study or future questions

- Stability: Can it only be done through adaptivity ?
- Least-Squares Space-time RBF-PU might be worth to try.
- Adaptivity in terms of partitions. Move away from points adaptivity.
- Preconditioner ?
- Possible GR application.
- Application to $2D + t$ Human Tear Film Dynamics.
- Enforce my grad students to finish the papers.


Fully open
2/3 open
1/3 open
Closed
x
y
x
y
x
y
x
y