Towards Effective Parallel Methods for Computing Pseudospectra

$\mathcal{PPP}$ Team:

- C. Bekas, E. Kokiopoulou, A. Sidiropoulos, EG (U. Patras)
- I. Koutis (Patras $\rightarrow$ Carnegie Mellon, Pitt)
- V. Simoncini (INA-CNR, Pavia)

Pseudospectrum of matrix A

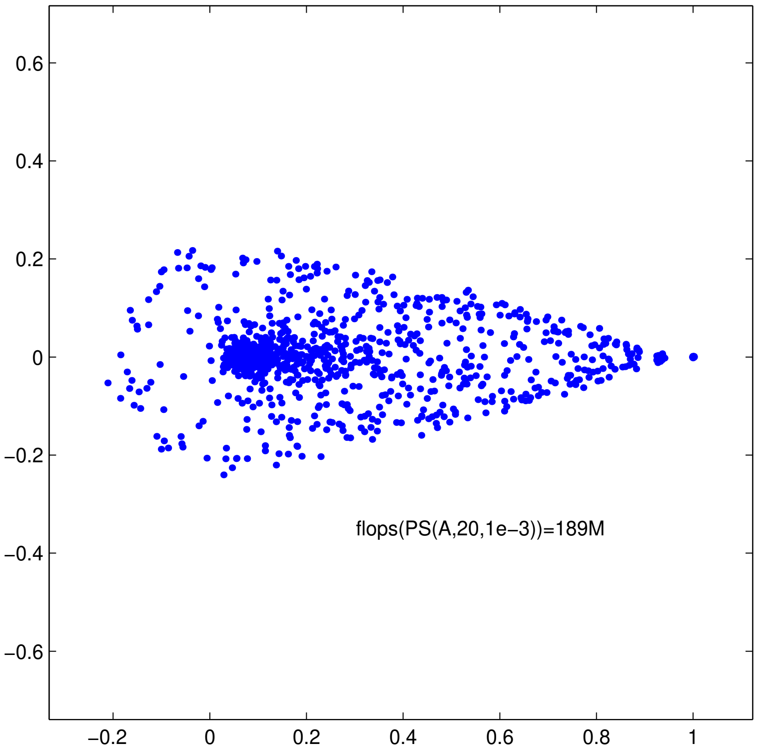
- $\Lambda_\epsilon(A) = \{z \in \mathbb{C} \ : \quad z \in \Lambda(A + \Delta A), \quad \|\Delta A\| \leq \epsilon\}$

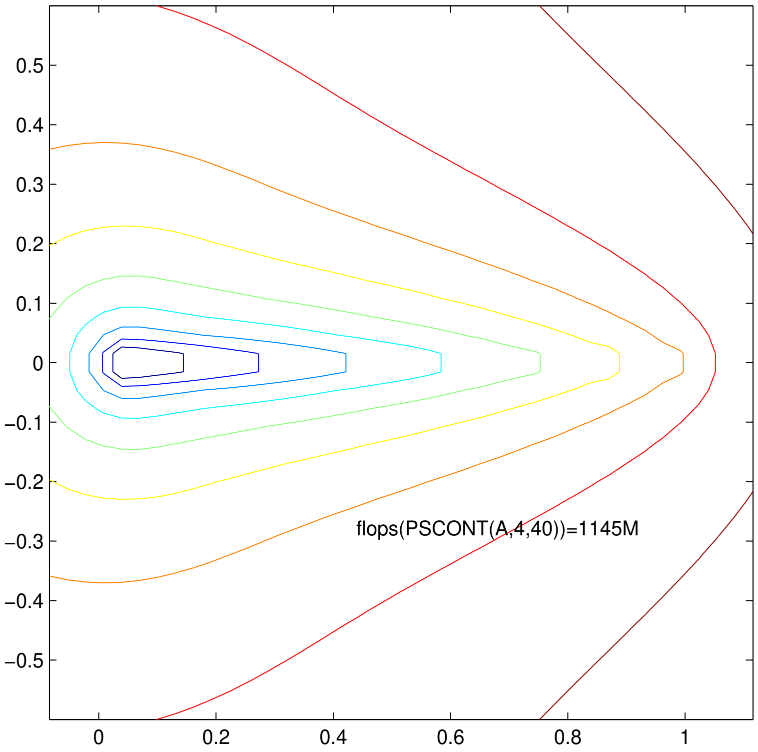
- $\Lambda_\epsilon(A) = \{z \in \mathbb{C} \ : \sigma_{\min}(zI - A) \leq \epsilon\}$

- $\Lambda_\epsilon(A) = \{z \in \mathbb{C} \ : \quad \|(zI - A)^{-1}\| \geq \epsilon^{-1}\}$

MATLAB TestMatrix tools

<u>**ps** and **pscont** applied on $A = \texttt{kahan}(50)$:</u>





Workhorse Method: `GRID`

$$\boxed{\begin{array}{c}\text{generate a grid}\\ \Downarrow \\ \text{perform SVD on every node}\\ \Downarrow \\ \text{plot contours}\end{array}}$$

Advantages

- Straightforward and robust
- Plot of $\Lambda_\epsilon(A)$ for several values of ϵ
- Independent calculations

BUT

$$\boxed{T(\texttt{GRID}) \approx \underbrace{|\Omega_h|}_{\text{number of points}} \times \underbrace{C_{\sigma_{\min}}}_{\text{avg. cost per point}}}$$

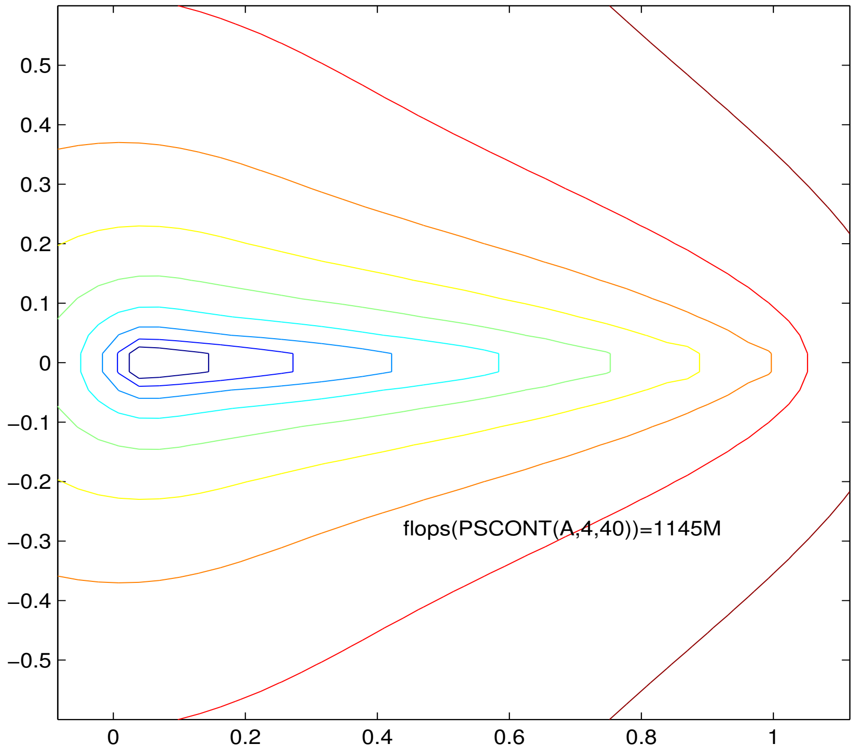
- Prohibitive cost for large matrices and/or fine grids

> *There is more to the computation of pseudospectra than the obvious* `GRID` *algorithm*

paraphrasing N. Trefethen,(*Acta Numerica'99*)

MATLAB TestMatrix toolbox `pscont`

II. `pscont` applied on $A = \texttt{kahan}(50)$ on 40×40 grid



Hierarchical Approach to reduce $Cost = |\Omega_h| \times C_{\sigma_{\min}}$

Domain level: Reduce the number of points at which the SVD is calculated.

- Path following, FOV tricks, SH algorithm
- Cobra, PAT (Mezher & Philippe'00), Inclusion-Exclusion/`MoG`, `PsDM`

Matrix level: Compress matrix and extract approximate SVD information.

- Arnoldi, Davidson, Lanczos (Philippe, ...)
- Transfer function framework (Simoncini & EG'98)

LAPACK kernel: Faster direct SVD, partial SVD

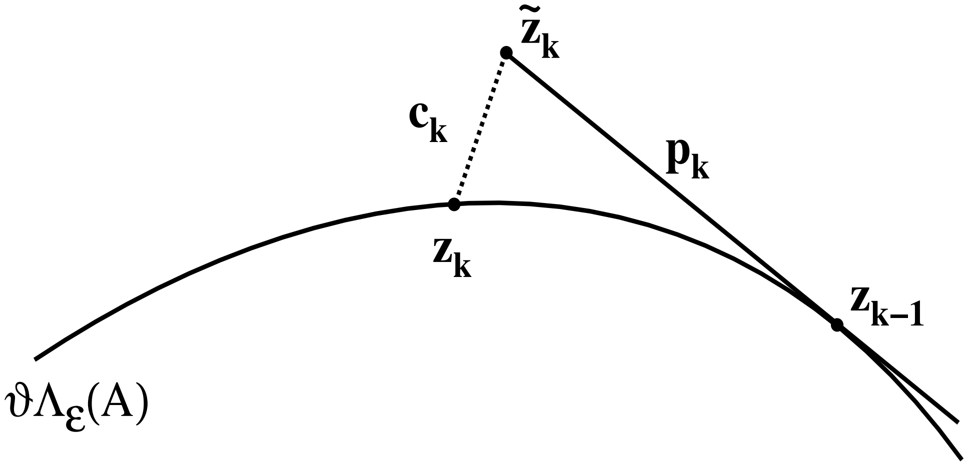
BLAS kernel: Optimizations

System: Parallel transformations

Engineering Tool

- user-friendly
  - widely available programming environments
  - widely available hardware
- fast
  - faster algorithms
  - parallel algorithms
- accurate, robust
- open source
- adaptable to various conditions
  - data
  - system

Pseudospectra with Path Following (Brühl'96)



FOR k=1,...

1.1 Determine a Prediction direction p_k.

1.2 Choose a steplength h and predict $\tilde{z}_k = z_{k-1} + hp_k$.

2.1 Determine a Correction direction c_k.

2.2 Correct $\tilde{z}_k$ to z_k using Newton on $\sigma_{\min}(zI - A) - \epsilon = 0$.

END

Known result on derivatives

$\partial\Lambda_\epsilon(A)$ is defined by the equation

$$\sigma_{\min}(zI - A) = \epsilon.$$

Theorem 1 *Let $z \in \mathbb{C} \setminus \Lambda(A)$. Then G(z) is real analytic in a neighborhood of z if $\sigma_{\min}(zI - A)$ is a simple singular value. Then the gradient of G at z is*

$$\nabla G(z) = (\Re(v_{\min}^* u_{\min}), \Im(v_{\min}^* u_{\min})) = v_{\min}^* u_{\min}, \tag{1}$$

where $v_{\min}, u_{\min}$ denote the left and right singular vectors corresponding to $\sigma_{\min}(zI - A)$.

Path following approach

$\surd$ **Path Following (`PF`)** traces the $\partial\Lambda_\epsilon(A)$ boundary

- $\surd$ drastically fewer SVD evaluations than `GRID`
- $\searrow$ lack of large grain parallelism
- $\searrow$ difficulties at regions of non-smooth $\partial\Lambda_\epsilon(A)$.
- $\searrow$ **single** component curve evaluated at each run
- $\searrow$ **disconnected components** not found with confidence

<u>**What can we do?**</u>

Cobra (BeGa'98-00 @ Copernicus/STABLE)

Three-phase path following w. multiple corrections

$\surd$ large grain parallelism

$\surd$ more robust

$\searrow$ limited scalability

$\searrow$ need several runs for different ϵ and disconnected components

What else?

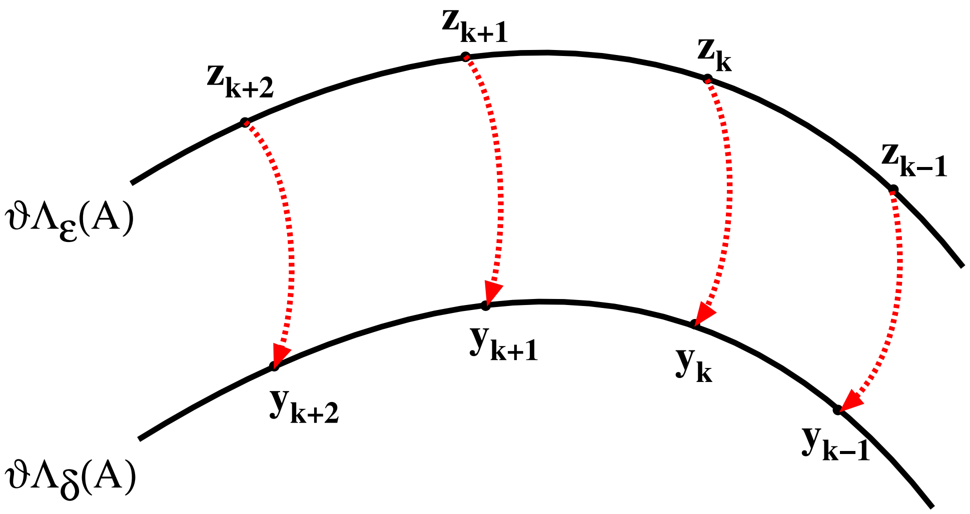
Obvious idea: $M > 1$ **curves** traced by running M (concurrent?) `PF` (`CPF`).

Idea: (Parallel) Pseudospectrum Descent Method (`PsDM`) (Bekas & EG'00)

Promising characteristics

- $\surd$ `PsDM` utilizes `PF` techniques to "correct" a point $z \in \partial\Lambda_{\delta_i}(A)$ to the point $y \in \partial\Lambda_{\delta_{i+1}}(A), \quad \delta_{i+1} < \delta_i$, towards the steepest descent direction.
- $\surd$ **Dynamic** adaptation to the length of the $\epsilon-$curve.
- $\surd$ **Disconnected** components can be captured.
- $\surd$ **Parallelism** opportunities such as `GRID`.

Descent Method (PsDM): Basic idea



√ **Goal**: Start from curve $\partial\Lambda_\epsilon(A)$ to estimate $\partial\Lambda_\delta(A)$, $\delta < \epsilon$.

√ **Points** z_k, $k = 1, \ldots$, resemble the points $\tilde{z}_k$, $k = 1, \ldots$, of `PF`.

√ **Thus**: estimate each y_k using Newton iteration on $\sigma_{\min}(u^i I - A) - \delta = 0$, $u^0 = z_k$.

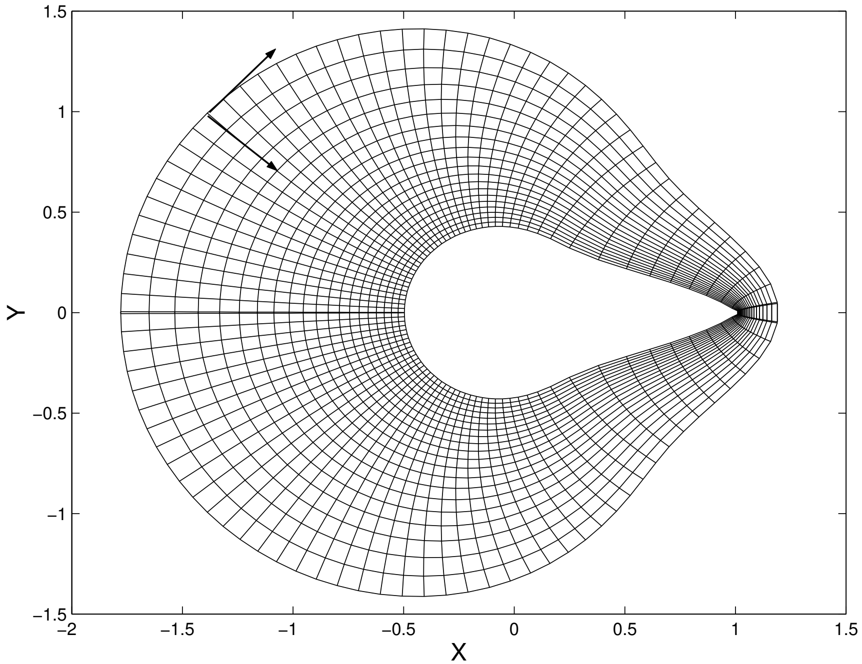
√ **Crucial**: Minimize singular triplet computation, use one Newton step as in `PF`.

Related $\mathcal{PPP}$ work: Spectrum from pseudospectrum (Koutis+G'99+)

Key:

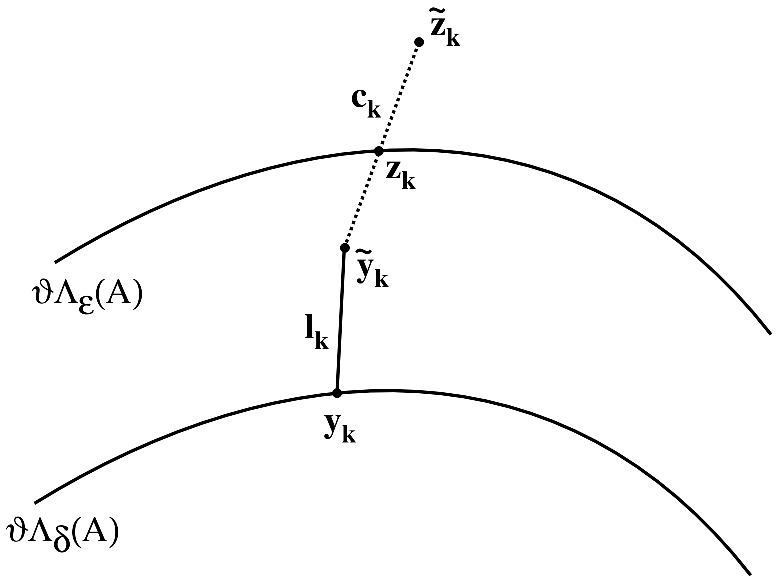
$$\Lambda(A) = \Lambda_0(A) = \{z|\sigma_{\min}(zI - A) = 0\}$$

- Gradient vectors point to zeroes of $\sigma_{\min}(zI - A) = 0$
- Starting from $z \in \Lambda_\epsilon(A)$ can compute elements of $\lambda \in \Lambda(A)$ using descent.



Pseudospectrum contours and trajectories of points computed by `PsDM` for $\partial\Lambda_\epsilon(A), \epsilon = 10^{-1}, ..., 10^{-3}$ for **`kahan(100)`**

PsDM: Detail of single step



For all points $z_k \in \partial\Lambda_\epsilon(A)$:

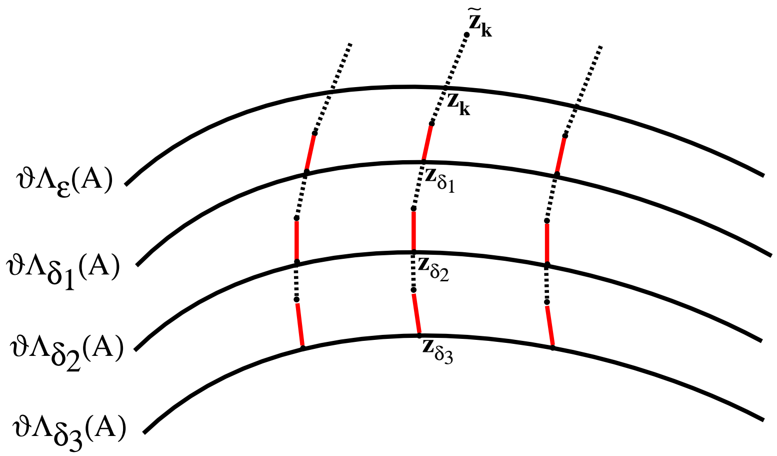
1. compute intermediate point $\tilde{y}_k = z_k - \frac{\epsilon - \delta}{(u^*_{\min} v_{\min})_{\tilde{z}_k}}$
2. compute new point $y_k = \tilde{y}_k - \frac{\sigma_{\min}(A - \tilde{y}_k I) - \delta}{(u^*_{\min} v_{\min})_{\tilde{y}_k}}$

B.Philippe equivalence with

$$\tilde{z}_k \overbrace{\longrightarrow}^{\text{Newton}} y_k \overbrace{\longrightarrow}^{\text{Newton}} y_k$$

for error analysis

`PsDM` method



Idea: Apply repeatedly single descent step

$\delta_0 = \epsilon$

FOR k=1,...

1. Select $\delta_k < \delta_{k-1}$
2. Apply one step of `PsDM` on $\partial\Lambda_{\delta_k}(A)$

END

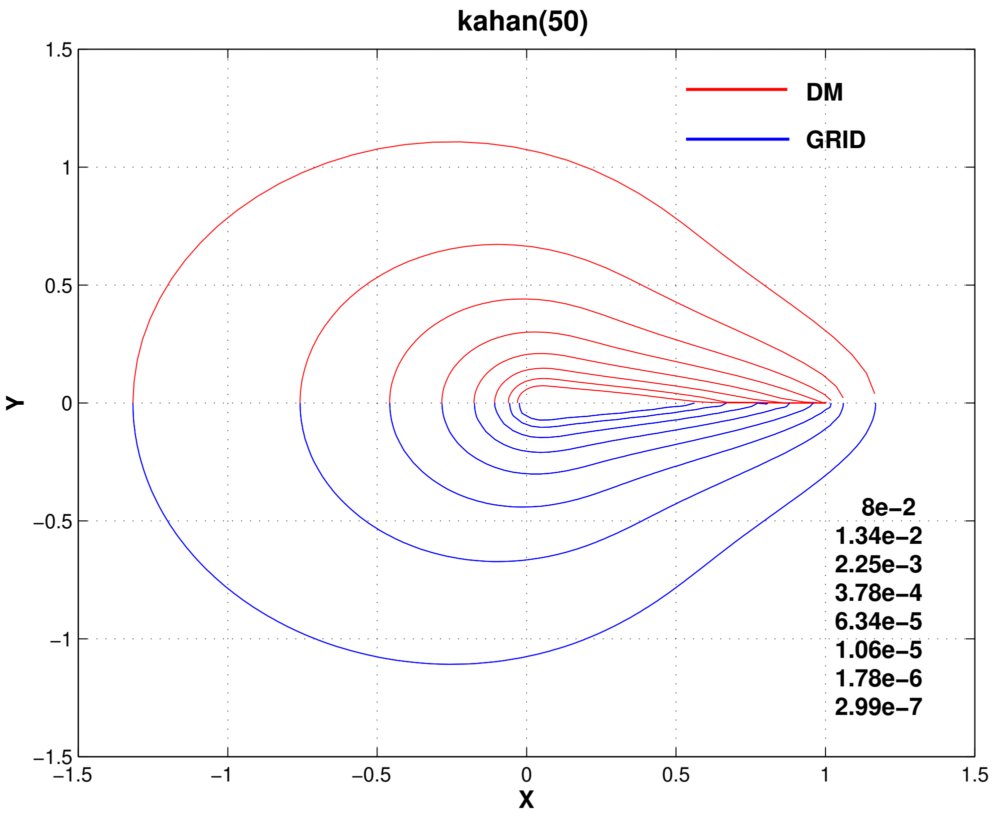
Numerical evidence

- **Celeron** 466 MHz, 128 MB RAM, `MATLAB` 5.3.
- **Start** `PsDM` from 40 points on $\partial\Lambda_{0.1}A$, for `kahan(50)`.
- **Computed** 60 curves, $\delta_{k+1} = 0.8\delta_k$, $\delta_{60} = 1.53 \times 10^{-7}$.

$\surd$ **performance:**

$$\begin{array}{lll} \texttt{DM}: & 2400\ \texttt{SVDs} & 80\,\mathrm{sec} \\ \texttt{GRID}: & 10^4 = 100 \times 100\ \texttt{SVDs} & 223\,\mathrm{sec} \end{array}.$$

Pictorial evidence



Parallel `PsDM`

*(*Input*)* N points z_k on the starting curve; P: number of processors

1. For each processor, select and assign the sweep computations for $\lceil N/P \rceil$ points.
2. Each processor proceeds with **`PsDM`** on its own set of points.
3. A marked processor gathers the results of all processors.

Parallel Performance

`SGI Origin 2000` with 8 `MIPS R10000`

`IRIX 6.5`

`F90/77 MIPSpro` version 7.2.1

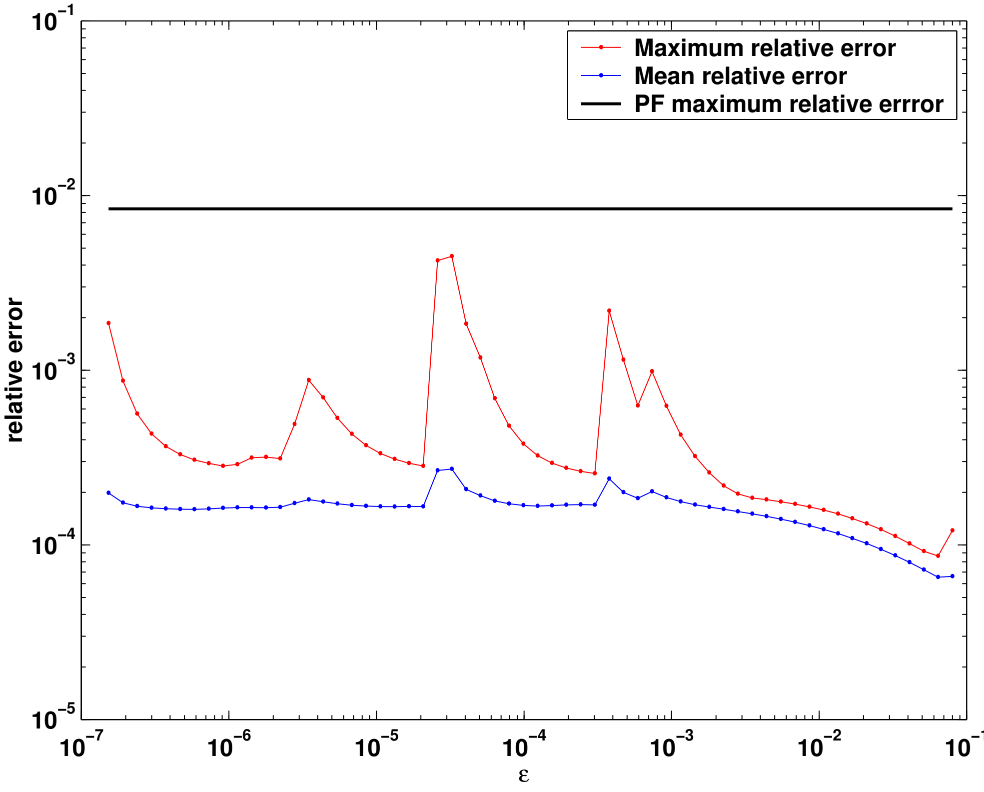
MPI programming paradigm, implemented by `SGI's MPT 1.2.1.0`

`ARPACK` and `SPARSKIT`

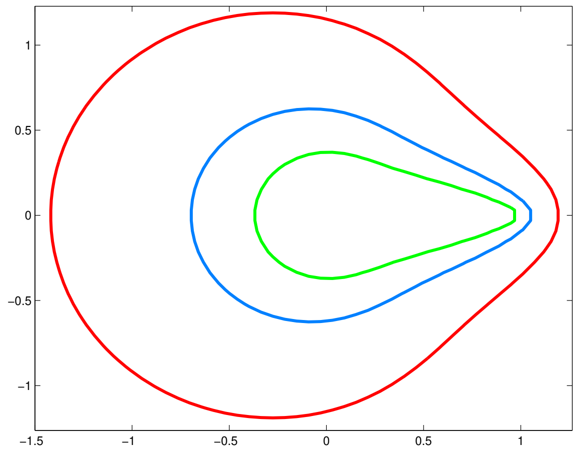
# of Processors	1	2	4	8
Time (secs)	264	135	70	36
Speedup	1	1.96	3.77	7.33

Performance of parallel implementation of `PsDM` for `kahan(100)`

Pictorial evidence

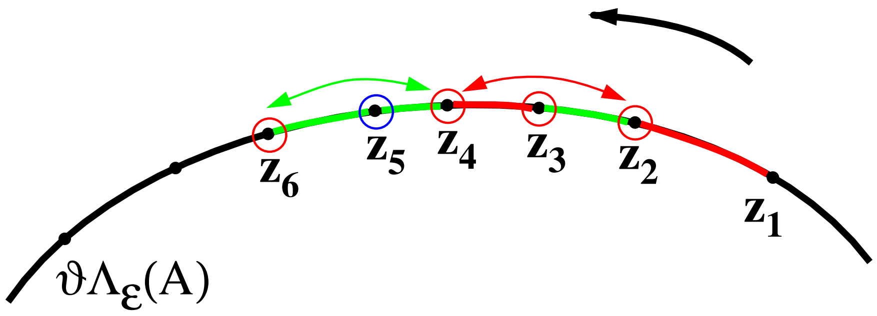


Exploiting shorter $\partial\Lambda_\delta(A)$



- $\surd$ **Example:** `kahan(50)`, $\epsilon = 10^{-1}, 10^{-2}, 10^{-3}$
- $\surd$ **Length** of curves may decrease significantly!
- $\surd$ **Propose to:** monitor length of computed curves and recalculate discretization.
- $\surd$ **Gain:** Possible drastic reduction of cost, since fewer SVDs are required.

Monitoring curve length: Mean value algorithm



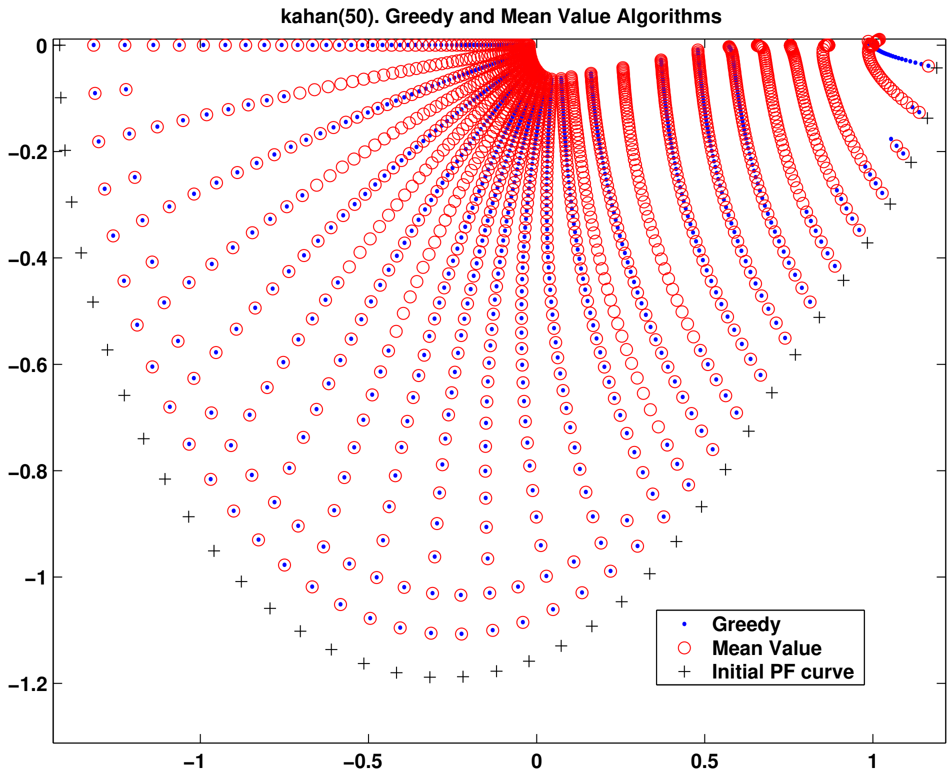
√ **Mean value algorithm**

*(*Input*)*

N_m points of $\partial\Lambda_{\delta_m}(A)$

1. Calculate $l_{\min} = \min |z_k - z_{k+1}|, z_k, z_{k+1} \in \partial\Lambda_{\delta_m}(A)$
2. Compute $\partial\Lambda_{\delta_{m+1}}(A)$ by `PsDM`
3. **FOR k=1,...,**$N_m - 2$, $\quad z_k \in \partial\Lambda_{\delta_{m+1}}(A)$
   3.1 $d_m = |z_k - z_{k+1}|, r_m = |z_{k+2} - z_{k+1}|$
   3.2 **IF** $d_m + r_m < 2 * l_m$ **DROP** z_{k+1}
   UNLESS z_k is **DROPPED**.
4. **END**

Exploiting shorter $\partial\Lambda_\delta(A)$.

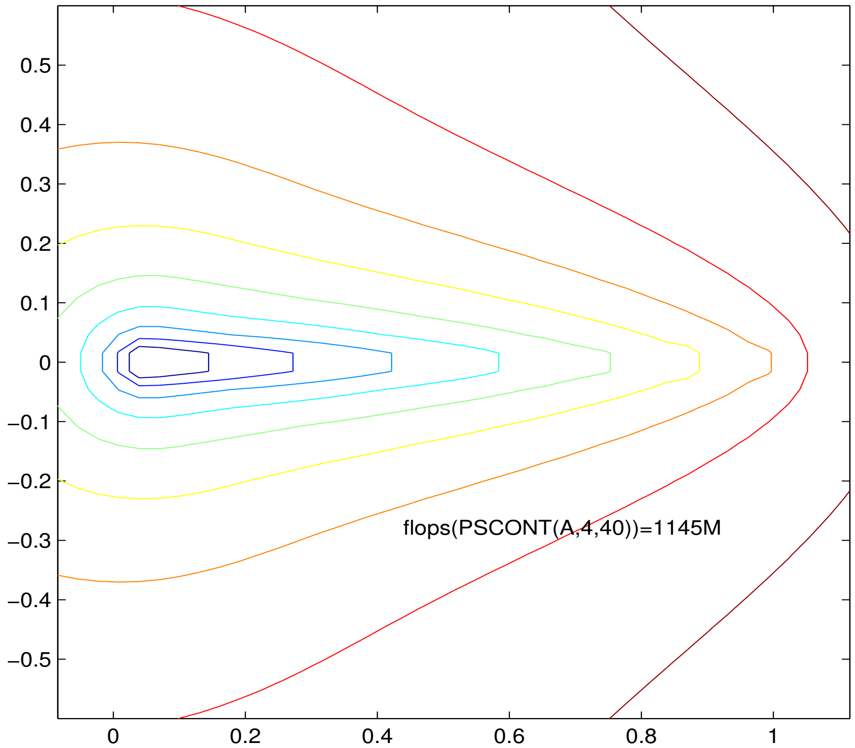


$\surd$ **PF:** 39 points $\partial\Lambda_{0.1}A$. **PsDM** $60 * 39 = 2340$ SVD's.

$\surd$ **Greedy:** 1051 SVD's, 2.3 times faster. **Mean Value:** 1400 SVD's, 1.7 times faster.

MATLAB TestMatrix toolbox `pscont`

II. `pscont` applied on $A = \texttt{kahan}(50)$ on 40×40 grid



Issues for `GRID`

$$\boxed{\begin{array}{c}\text{generate grid } \Omega_h \\ \Downarrow \\ \text{perform SVD on every gridpoint} \\ \Downarrow \\ \text{plot contours}\end{array}}$$

$$\Downarrow$$

Where to place Ω_h?

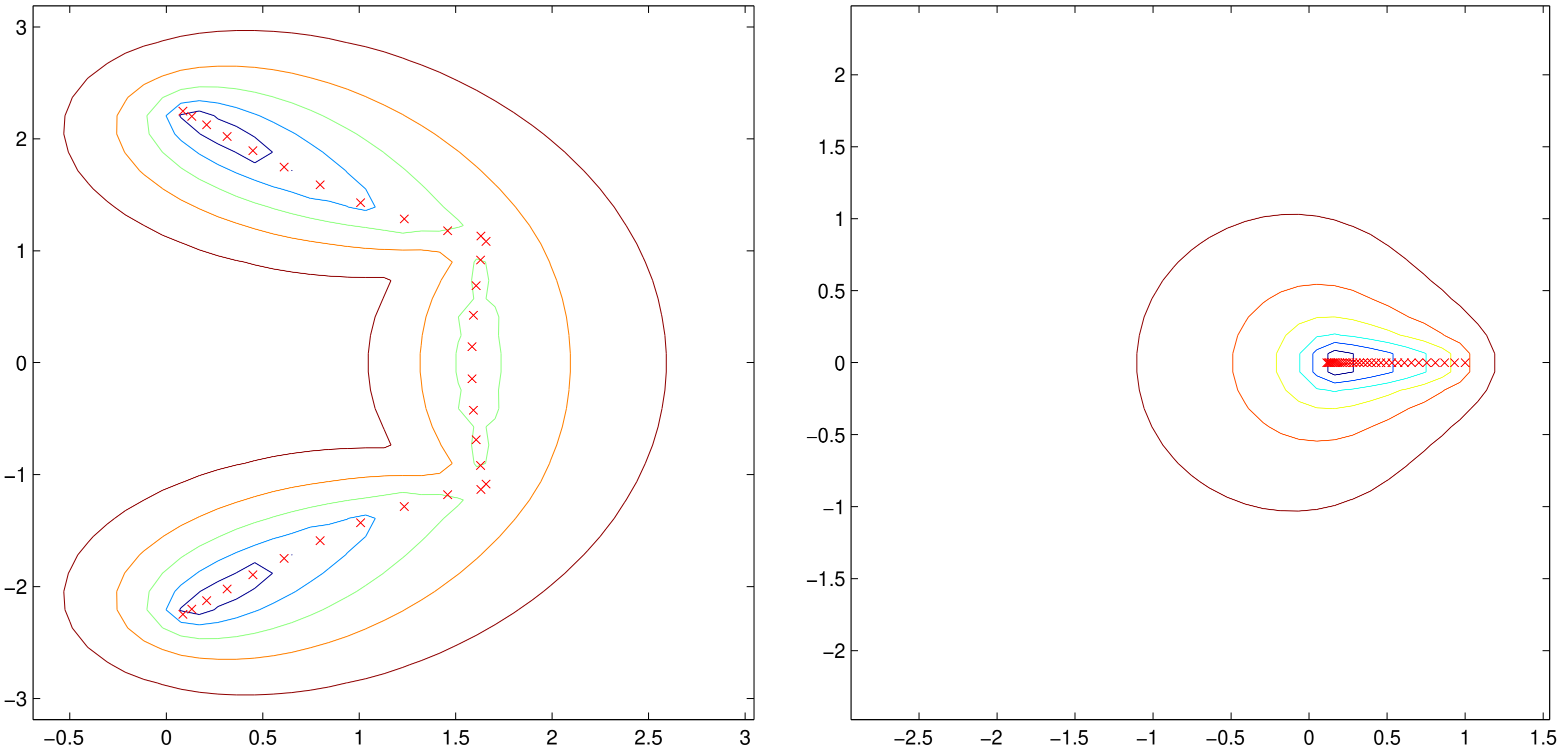
Comments

- Most Ω estimates do not fully enclose $\Lambda_\epsilon(A)$.
- Rectify by overestimating

  The larger the overestimate the larger the cost

Possible overestimates due to:

- Enclosing region region too large;
- Enclosing region of unsuitable shape
  - e.g. convex Ω to approximate nonconvex $\Lambda_\epsilon(A)$



Pseudospectrum contours, computed using **`pscont`** and a field of values criterion for matrices **`grcar`** (left) and **`kahan`** (right) of size $n = 32$ and $\epsilon = 1e-9, ..., 1e-1$.

Exclusion strategy:

- *Prune* Ω from subdomains known to be disjoint from Λ_ϵ.

General Inclusion-Exclusion method

1. Obtain inclusion region $\Omega \supset \Lambda_\epsilon(A)$.
2. Compute exclusion regions Δ_j such that:

$$\Omega \cap \Delta_j \neq \emptyset \quad \textbf{and} \quad \Lambda_\epsilon(A) \cap \Delta_j = \emptyset$$

   and set

$$\Omega = \Omega \backslash \cup_{j=1:n} \Delta_j$$

3. Discretize with grid $\Omega_h \subset \Omega$.
4. Compute singular triplets on Ω_h.

GOAL: Obtain effective exclusion regions

Recent tools (Koutis, G.'99)

Theorem 2 (Exclusion)

If $\sigma_{\min}(zI - A) = r > \epsilon$ *then*[a] $D^\circ(z, r-\epsilon) \cap \Lambda_\epsilon(A) = \emptyset$.

[a] $D^\circ(z, r)$ denotes the open disk.

Assume $z \in \Omega_h$ and $\sigma_{\min}(zI - A) = r > \epsilon$.

Then we can exclude from Ω_h all points in $D(z, r-\epsilon)$.

Approach: Algorithm `MoGO`

Use `MoGO` *method with systematic application of Exclusion Theorem 2*

Savings?

Compare

Grid is pointwise: For every point z, the computation of $\sigma_{\min}(z)$ tells us to exclude or not that point.

$\Downarrow$

Have to look at all the gridpoints

`MoG` acts at a distance: For every point z, the computation of $\sigma_{\min}(z)$ might lead to exclusions of nearby grid points

$\Downarrow$

Look only at subset of grid

Enhancements: Known result on derivatives

$\partial\Lambda_\epsilon(A)$ is defined by the equation

$$\sigma_{\min}(zI - A) = \epsilon.$$

Theorem 3 (Sun) *Let $z = x + iy \in \mathbb{C} \setminus \Lambda(A)$, $\sigma_{\min}(zI - A)$ simple and $u_{\min}, v_{\min}$ the singular vectors.*
Then $g_A(z) := \sigma_{\min}(zI - A)$ is differentiable in a neighborhood of z and

$$\nabla g_A(x,y) = (Re(v_{\min}^* u_{\min}), Im(v_{\min}^* u_{\min})) = v_{\min}^* u_{\min}.$$

Define

$$\mathrm{pss}_A(z_0) := [|\nabla g_A(z)|]^{-1}$$

as local sensitivity of the pseudospectrum at z_0.

Key II

Theorem [Koutis+G.99]

Under some appropriate conditions that typically hold, and

$$R = ((g_A(z_0) - \epsilon))\mathrm{pss}_A(z_0)$$

then $D^o(z, R) \cap \Lambda_\epsilon(A) = \emptyset$.

$$g_A(z_0) - \epsilon \leq \overbrace{((g_A(z_0) - \epsilon))\mathrm{pss}_A(z_0)}^{R}$$
$$\Downarrow$$
$$D^\circ(z_0, g_A(z_0) - \epsilon) \subseteq D^\circ(z_0, R)$$

More effective exclusions

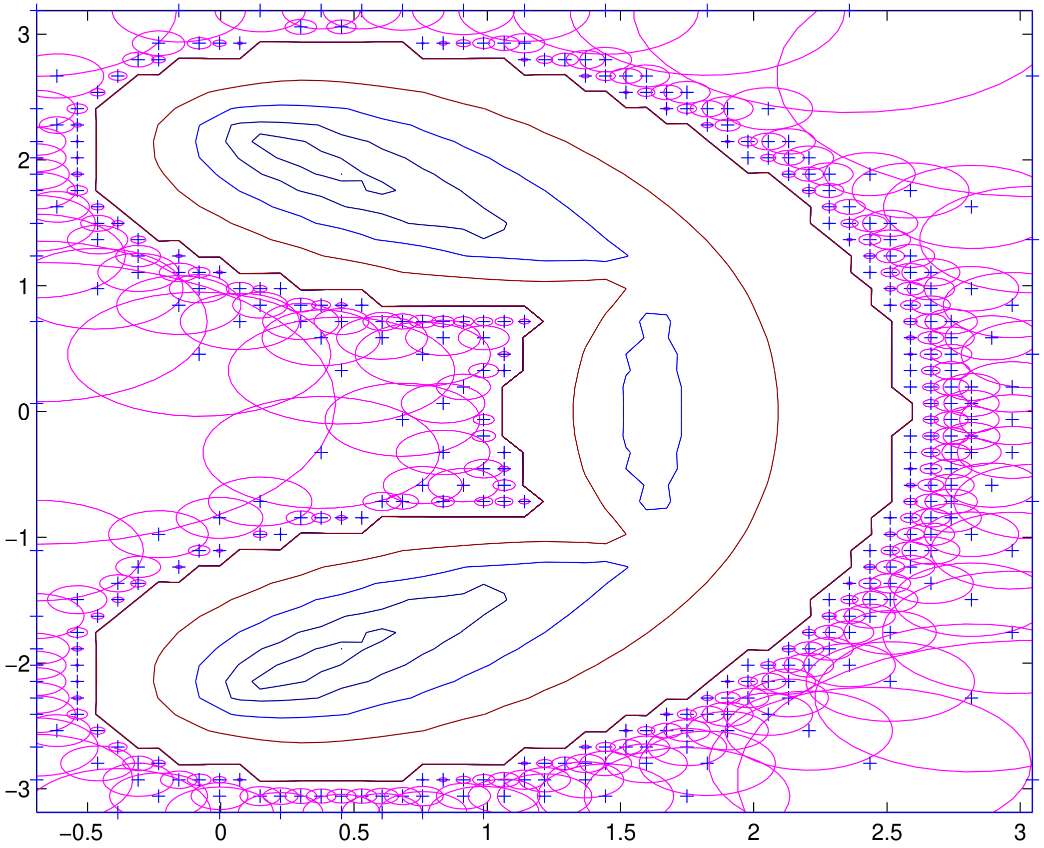
Approach: Algorithm `MoG`

Systematic application of Exclusion Theorem **??** *to exclude regions from inside the enclosing region*

The `MoG` algorithm

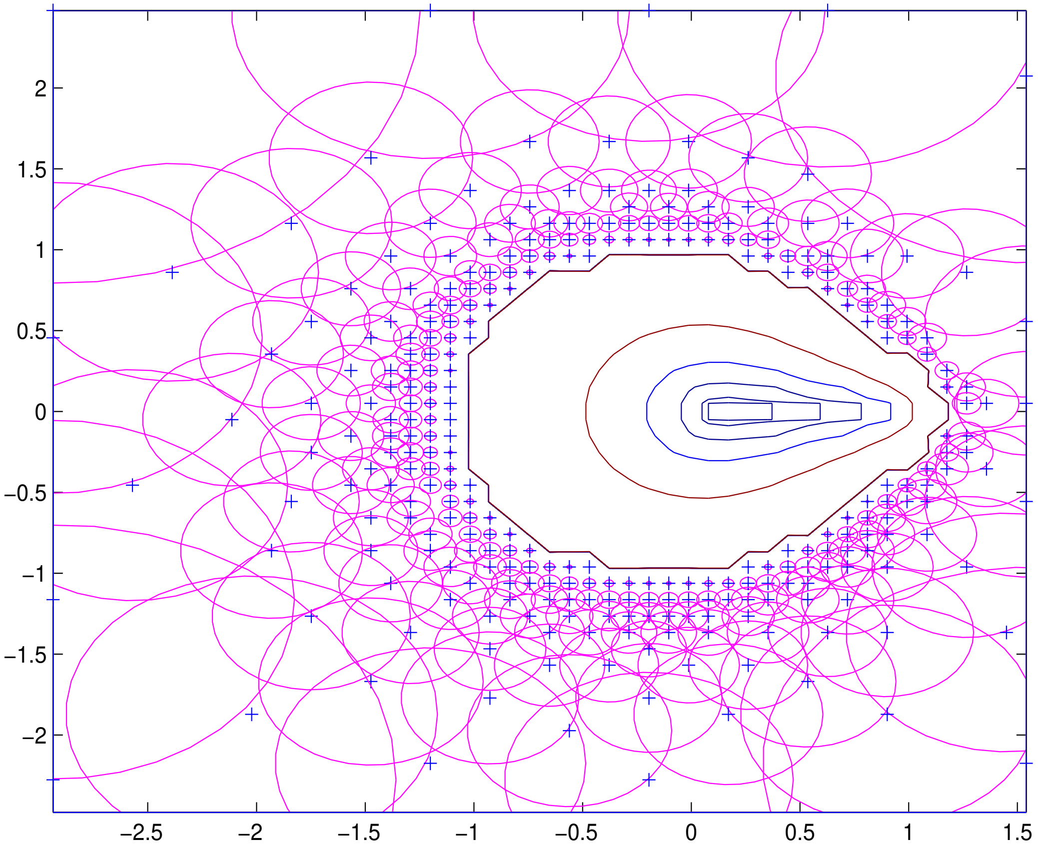
1. choose enclosing region $\Omega \supseteq \Lambda_\epsilon(A)$,
discretize it with Ω_h and mark all points of Ω_h as “`un`”.
2. **for** every grid point $z \in \Omega_h$
compute $\sigma = \sigma_{\min}(zI - A)$.
if $\sigma > \epsilon$ then
case `mmx = 0`: mark all points in $D^\circ(z, \sigma - \epsilon)$ as “`out`”
and set $\Omega_h = \Omega_h \setminus D^\circ(z, \sigma - \epsilon)$
case `mmx = 1`: mark all points in $D^\circ(z, pss(z)(\sigma - \epsilon))$ as “`out`”
and set $\Omega_h = \Omega_h \setminus D^\circ(z, pss(z)(\sigma - \epsilon))$.
Apply `proc_corr_A`
else mark z as “`in`”
end for
3. **if** `mmx = 1` apply `proc_corr_B`.
4. Draw contours for the selected values of ϵ at all points marked as “`in`”.

`MoGO` **for** `grcar(32)` **and** $\epsilon = 1e-1$



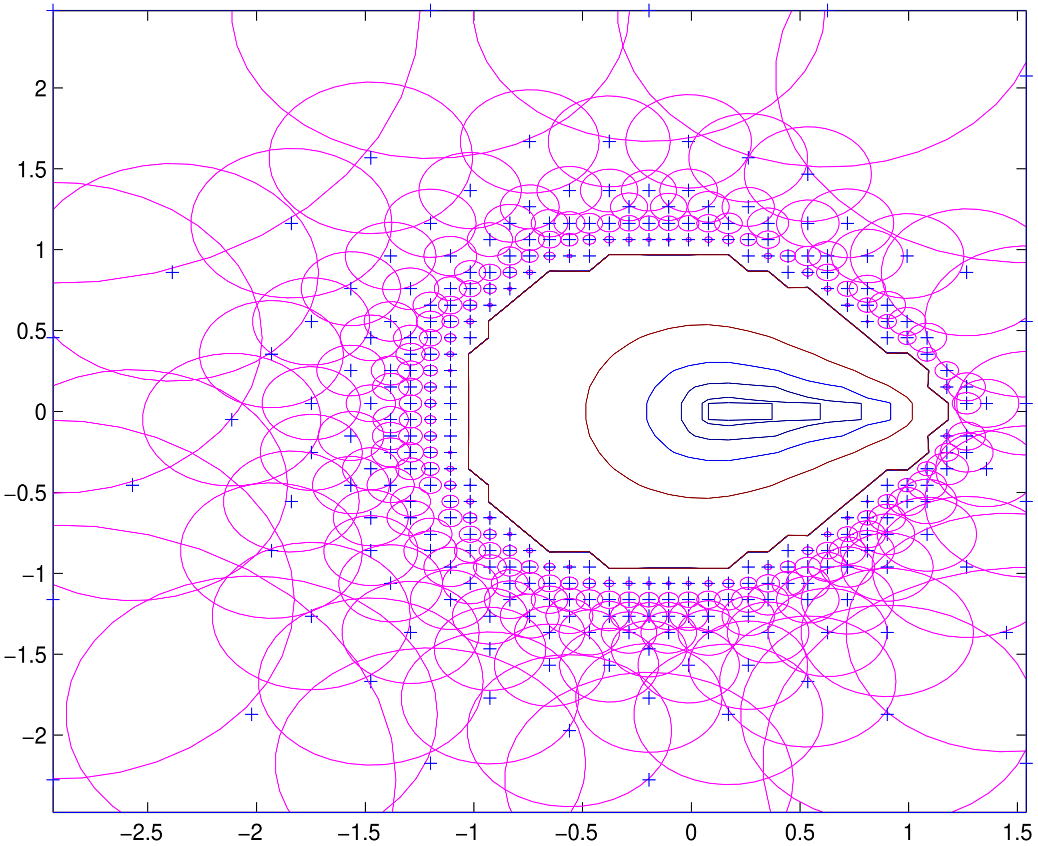
Application of **`MoGO`** using 50×50 grid, totaled 1564 triplet calculations.

`MoGO` **for** `kahan(32)` **and** $\epsilon = 1e-1$



Application of `MoGO` using 50×50 grid totaled 667 triplet calculations.

`MoG` for `kahan(32)` and $\epsilon = 1e-1$



Application of `MoG` using 50×50 grid totaled 499 triplet calculations.

Numerical Experiments

matrix	region Ω	`MoGO`	`MoG`
`grcar(n)`	$[-0.69, 3.05] \times [-3.19, 3.19]$	790	738
`kahan(n)`	$[-2.94, 1.54] \times [-2.48, 2.48]$	351	267
`chebspec(n,0)`	$[-375.60, 375.60] \times [-264.75, 264.75]$	311	129
`chebspec(n,1)`	$[-400.20, 108.43] \times [-281.30, 281.30]$	225	91
`chow(n)`	$[-0.60, 17.57] \times [-10.18, 10.18]$	343	218
`triangle(n)`	$[-0.85, 1.34] \times [-1.19, 1.19]$	865	847
`propeller(n)`	$[-1.12, 1.58] \times [-1.45, 1.45]$	846	823
`fish(n)`	$[-0.37, 3.58] \times [-1.45, 1.45]$	962	931

Number of $\sigma_{\min}$ evaluations in a 50×50 grid using the two versions of algorithm `MoG` for maximum $\epsilon = 1e - 1$ and $n = 32$.

Compare with 2500 SVD's for standard GRID

matrix	region Ω	`MoGO`	`MoG`
`grcar(n)`	$[-0.59, 2.95] \times [-3.09, 3.09]$	692	403
`kahan(n)`	$[-2.84, 1.44] \times [-2.38, 2.38]$	265	114
`chebspec(n,0)`	$[-375.50, 375.50] \times [-264.65, 264.65]$	312	121
`chebspec(n,1)`	$[-400.10, 108.33] \times [-281.20, 281.20]$	225	81
`chow(n)`	$[-0.50, 17.47] \times [-10.09, 10.09]$	321	134
`dramadah(n)`	$[-1.51, 10.15] \times [-4.96, 4.96]$	421	129
`triangle(n)`	$[-0.75, 1.24] \times [-1.09, 1.09]$	777	638
`propeller(n)`	$[-1.02, 1.48] \times [-1.35, 1.35]$	754	570
`fish(n)`	$[-0.27, 3.48] \times [-1.35, 1.35]$	896	682

Number of $\sigma_{\min}$ evaluations in a 50×50 grid using the two versions of algorithm `MoG` for maximum $\epsilon = 1e - 3$ and $n = 32$.

Numerical Experiments

matrix	size n	region Ω	`MoG0`	`MoG`
`kahan(32)`	70	$[-2.84, 1.44] \times [-2.38, 2.38]$	426	167
	90		613	226
`triangle(32)`	70	$[-0.75, 1.24] \times [-1.09, 1.09]$	1442	1112
	90		2295	1697
`propeller(32)`	70	$[-1.02, 1.48] \times [-1.35, 1.35]$	1387	960
	90		2188	1435

Number of $\sigma_{\min}$ evaluations for varying grid sizes using two versions of algorithm `MoG` and maximum $\epsilon = 1e-3$.

Compare with 4900 and 8100 SVD's for standard GRID

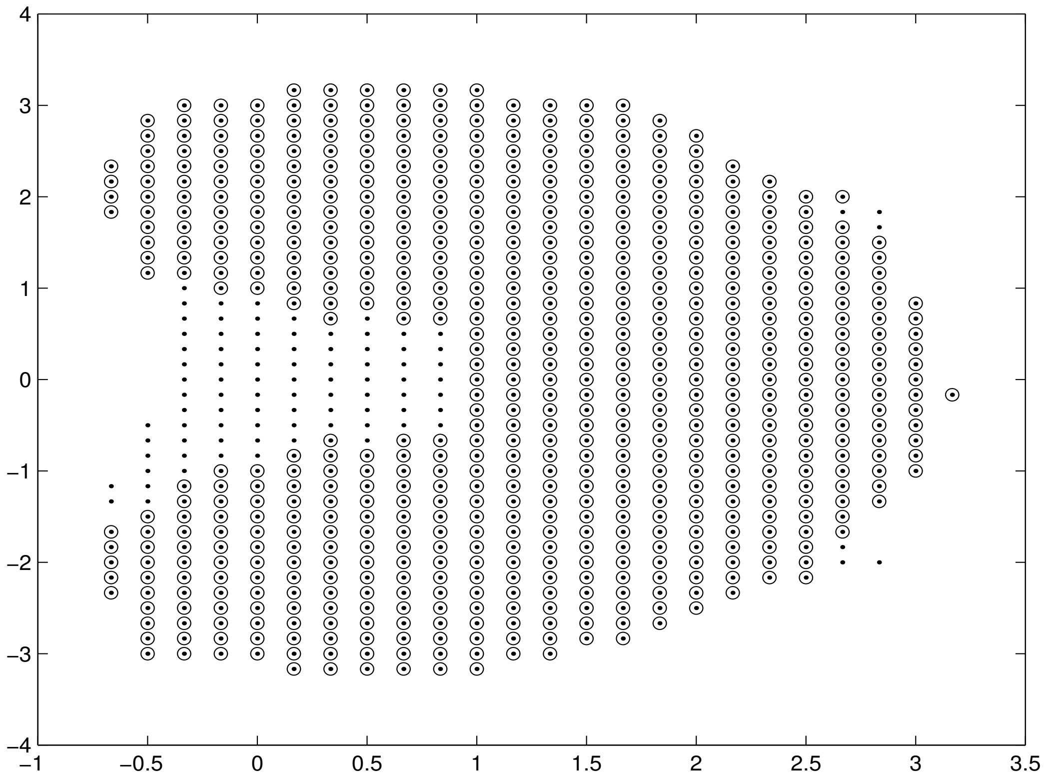
Java Implementation: (A. Sidiropoulos'01)

Very interesting parallelization issues (Bekas,Kokiopoulou,Koutis+EG'01)

- Exclusions depend on the order of the sweep
- Reverse Locality: If a point z is excluded then those near it are also likely to be excluded!
- Three-phase algorithm on network of 16 PC's running Windows
- MATLAB Multitask Toolbox over MPI

$$1) \text{ Exclude} \Rightarrow 2) \text{ Collapse} \Rightarrow 3) \text{ Grid}$$

Example for `grcar(1000)` on 50×50 grid



Results

P	Phase1	Phase 2	SB		MC		QUEUE	
			Ph. 3	sum	Ph. 3	sum	Ph. 3	sum
1	794	1324	4756	6874	-	-	-	-
2	574	644	2290	3508	2436	3654	2132	3350
4	271	347	1216	1834	1205	1823	1072	1690
8	183	158	540	881	576	917	523	864
15	102	93	299	494	300	495	280	475
16	81	104	275	460	278	463	-	-

Runtimes (sec) for `MoG` on 10^{-7}`pores_2`. $\Omega = [-2.5, 1.5] \times [-1.5, 1.5]$, 50×50 grid, $\epsilon = 0.1$