The Sparse Grids Matlab Kit

C. Piazzola, L.Tamellini[1]

[1] CNR-IMATI, Pavia

February 25, 2022

Outline

Contributors, releases

- main contributors: Lorenzo Tamellini, Chiara Piazzola

Contributors, releases

- main contributors: Lorenzo Tamellini, Chiara Piazzola
- other contributors: Fabio Nobile, Eva Vidlickova (EPFL), Diane Guignard (U. Ottawa), Giovanni Porta (Politecnico Milano), Björn Sprungk (TU Freiberg)

Contributors, releases

- main contributors: Lorenzo Tamellini, Chiara Piazzola
- other contributors: Fabio Nobile, Eva Vidlickova (EPFL), Diane Guignard (U. Ottawa), Giovanni Porta (Politecnico Milano), Björn Sprungk (TU Freiberg)
- releases:

Contributors, releases

- main contributors: Lorenzo Tamellini, Chiara Piazzola
- other contributors: Fabio Nobile, Eva Vidlickova (EPFL), Diane Guignard (U. Ottawa), Giovanni Porta (Politecnico Milano), Björn Sprungk (TU Freiberg)
- releases:
  - 22-2 (“California”)
  - 18-10 (“Esperanza”)
  - 17-5 (“Trent”)
  - 15-8 (“Woodstock”)
  - 14-12 (“Fenice”)
  - 14-4 (“Ritchie”)

Contributors, releases

- main contributors: Lorenzo Tamellini, Chiara Piazzola
- other contributors: Fabio Nobile, Eva Vidlickova (EPFL), Diane Guignard (U. Ottawa), Giovanni Porta (Politecnico Milano), Björn Sprungk (TU Freiberg)
- releases:
  - 22-2 ("California")
  - 18-10 ("Esperanza")
  - 17-5 ("Trent")
  - 15-8 ("Woodstock")
  - 14-12 ("Fenice")
  - 14-4 ("Ritchie")
- Download from `https://sites.google.com/view/sparse-grids-kit`

Contributors, releases

- main contributors: Lorenzo Tamellini, Chiara Piazzola
- other contributors: Fabio Nobile, Eva Vidlickova (EPFL), Diane Guignard (U. Ottawa), Giovanni Porta (Politecnico Milano), Björn Sprungk (TU Freiberg)
- releases:
  - 22-2 ("California")
  - 18-10 ("Esperanza")
  - 17-5 ("Trent")
  - 15-8 ("Woodstock")
  - 14-12 ("Fenice")
  - 14-4 ("Ritchie")
- Download from `https://sites.google.com/view/sparse-grids-kit`
- BSD2 license

General comments

- Lightweight, high-level and (hopefully) easy to use, good for quick prototyping and teaching

General comments

- Lightweight, high-level and (hopefully) easy to use, good for quick prototyping and teaching
- Some ad-hoc features and integration with parallel toolbox gives nonetheless reasonable speed

General comments

- Lightweight, high-level and (hopefully) easy to use, good for quick prototyping and teaching
- Some ad-hoc features and integration with parallel toolbox gives nonetheless reasonable speed
- Very extensive documentation and examples (8800 lines of code, 4800 lines of comments)

General comments

- Lightweight, high-level and (hopefully) easy to use, good for quick prototyping and teaching
- Some ad-hoc features and integration with parallel toolbox gives nonetheless reasonable speed
- Very extensive documentation and examples (8800 lines of code, 4800 lines of comments)
- Geared towards UQ, but flexible enough for other purposes

General comments

- Lightweight, high-level and (hopefully) easy to use, good for quick prototyping and teaching
- Some ad-hoc features and integration with parallel toolbox gives nonetheless reasonable speed
- Very extensive documentation and examples (8800 lines of code, 4800 lines of comments)
- Geared towards UQ, but flexible enough for other purposes
- Similar to:
  - Dakota, PyApprox, MUQ, ChaosPy, UQTk (not Matlab);
  - SG++, Tasmanian (Matlab wrappers);
  - UQLab, Spinterp (Matlab).

General comments

- Lightweight, high-level and (hopefully) easy to use, good for quick prototyping and teaching
- Some ad-hoc features and integration with parallel toolbox gives nonetheless reasonable speed
- Very extensive documentation and examples (8800 lines of code, 4800 lines of comments)
- Geared towards UQ, but flexible enough for other purposes
- Similar to:
  - Dakota, PyApprox, MUQ, ChaosPy, UQTk (not Matlab);
  - SG++, Tasmanian (Matlab wrappers);
  - UQLab, Spinterp (Matlab).

Aim of these slides: give rough idea of sparse grids and of the structure of the code, show features by examples

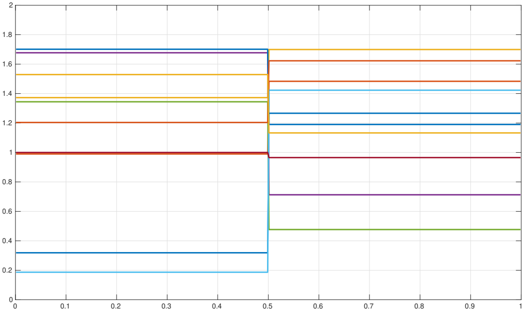
Outline

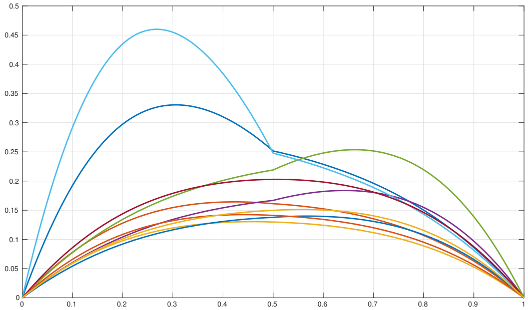
Forward Uncertainty Quantification

$$
\begin{cases}
-\dfrac{d}{dx}\left[a(x,\boldsymbol{y})\dfrac{d}{dx}u(x,\boldsymbol{y})\right] = 1, \quad \text{for } x \in (0,1) \\
u(0,\cdot) = u(1,\cdot) = 0 \\
a(x,\boldsymbol{y}) = 1 + \sigma_1 y_1 \mathbb{I}_{[0,0.5]}(x) + \sigma_2 y_2 \mathbb{I}_{[0.5,1]}(x) \\
{\color{red} y_1, y_2 \sim \mathcal{U}(-\sqrt{3}, \sqrt{3})} \\
I(\boldsymbol{y}) = \int_0^1 u(x,\boldsymbol{y})dx
\end{cases}
$$

- often, more than $N = 2$ parameters
- ODE and algebraic models are also of interest

Forward Uncertainty Quantification





Forward Uncertainty Quantification

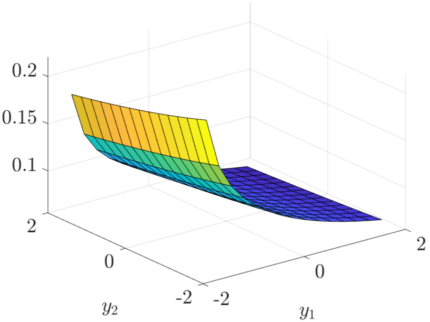
Goals:

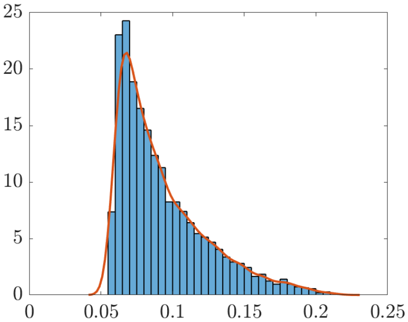
- statistics of I (mean, variance,pdf)
- sensitivity analysis for I
- meta-model for $I(y_1, y_2)$ (aka response surface aka surrogate model)

Need:

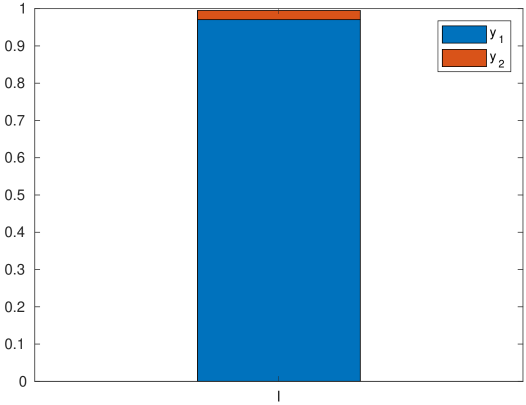
- High-dimensional quadrature (mean, variance)
- High-dimensional interpolation/approximation (pdf, metamodel)

Results

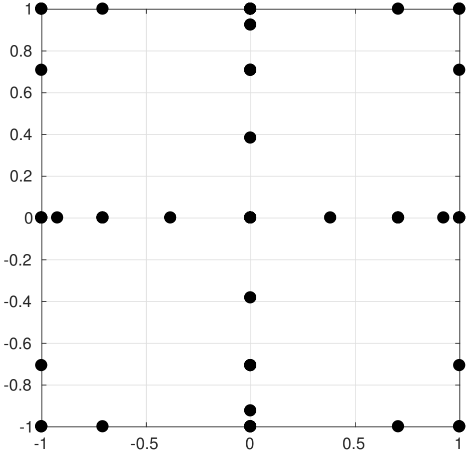




Results



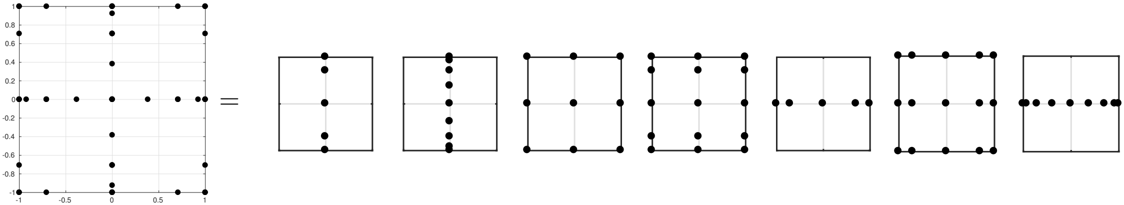
Where do we sample? Sparse grids



Our matlab library handles:

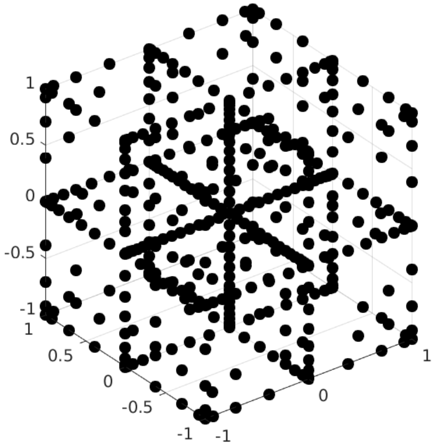
- generation of these schemes
- integration and interpolation on them

Sparse grids in a nutshell



- Linear combination of tensor grids
- Reasonably slows down "curse of dimensionality" up to N =a few tens
- Works well under regularity assumptions on $I(\boldsymbol{y})$
- Anisotropic and adaptive versions available

Sparse grids in a nutshell



Outline

Backbone: combination technique & "reduced" sparse grid

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

Backbone: combination technique & “reduced” sparse grid

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- $S = \sum_{\mathrm{i} \in \mathcal{I}} c_{\mathrm{i}} \otimes_{n=1}^{N} \mathcal{U}^{m(i_n)}$

Backbone: combination technique & “reduced” sparse grid

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- $S = \sum_{\mathrm{i}\in\mathcal{I}} c_{\mathrm{i}} \otimes_{n=1}^{N} \mathcal{U}^{m(i_n)}$
- $m(i) =$ “$2^{i-1}+1$”, $\quad \mathcal{U}^{m(i_n)} =$ interpolant on $m(i_n)$ Clenshaw–Cts pts

Backbone: combination technique & “reduced” sparse grid

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- $S = \sum_{\mathrm{i} \in \mathcal{I}} c_{\mathrm{i}} \otimes_{n=1}^{N} \mathcal{U}^{m(i_n)}$
- $m(i) =$“$2^{i-1} + 1$”, $\quad \mathcal{U}^{m(i_n)} =$ interpolant on $m(i_n)$ Clenshaw–Cts pts
- $\mathcal{I} = \left\{ \mathrm{i} \in \mathbb{N}_+^N : \sum_{n=1}^{N} (i_n - 1) \leq w \right\}$

Backbone: combination technique & “reduced” sparse grid

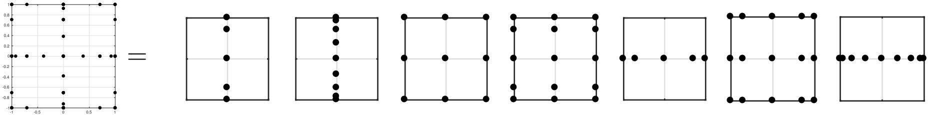
```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

```
>> S
    S =
    1 x 7 struct array with fields:
    knots
    weights
    size
    knots_per_dim
    m
    coeff
    idx
```

Backbone: combination technique & “reduced” sparse grid

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

```
>> S
    S =
    1 x 7 struct array with fields:
    knots
    weights
    size
    knots_per_dim
    m
    coeff
    idx
```



Backbone: combination technique & “reduced” sparse grid

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

```
>> S(1)
ans =
struct with fields:

           knots: [2x5 double] % points are always column vectors
         weights: [-0.1333 -1.0667 -1.6000 -1.0667 -0.1333]
            size: 5
   knots_per_dim: {[0]  [1 0.7071 6.1232e-17 -0.7071 -1]}
               m: [1 5]
           coeff: -1
             idx: [1 3] %multiidx are always row vectors
```

Backbone: combination technique & “reduced” sparse grid

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

```
>> Sr = reduce_sparse_grid(S)
Sr =

  struct with fields:

      knots: [2x29 double]
          m: [29x1 double]
    weights: [1x29 double]
          n: [67x1 double]
       size: 29
```

Backbone: combination technique & “reduced” sparse grid

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

```
>> Sr = reduce_sparse_grid(S)    % creates the "uniqued" list of points
                        % from the set of points of all stensor grids in S.
                        % Robust to numerical noise (default tol 1e-14)
Sr =

  struct with fields:

      knots: [2x29 double]
          m: [29x1 double] % map from Sr.knots to [S.knots]
    weights: [1x29 double]
          n: [67x1 double] % map from [S.knots] to Sr.knots
       size: 29            % nb of points in the sparse grids
```

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

	knots	**function**	**nestedness**
uniform	Gauss–Legendre	`knots_uniform`	No
	Clenshaw–Curtis	`knots_CC`	Yes
	Leja (standard, symmetric, p-disk)	`knots_leja`	Yes
	midpoints	`knots_midpoint`	Yes
	equispaced	`knots_trap`	Yes
normal	Gauss–Hermite	`knots_normal`	No
	Genz–Keister	`knots_GK`	Yes
	weighted Leja (standard, symmetric)	`knots_normal_leja`	Yes
exponential	Gauss–Laguerre	`knots_exponential`	No
	weighted Leja	`knots_exponential_leja`	Yes
gamma	Gauss–generalized Laguerre	`knots_gamma`	No
	weighted Leja	`knots_gamma_leja`	Yes
beta	Gauss–Jacobi	`knots_beta`	No
	weighted Leja (standard, symmetric)	`knots_beta_leja`	Yes

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- **`m = lev2knots_lin(i)`**

  $m(i) = i$

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- `m = lev2knots_lin(i)`
- `m = lev2knots_2step(i)`

  $m(i) = 2(i-1)+1$

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- `m = lev2knots_lin(i)`
- `m = lev2knots_2step(i)`
- `m = lev2knots_doubling(i)`
  $m(1) = 1, m(i) = 2^{i-1} + 1$

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- `m = lev2knots_lin(i)`
- `m = lev2knots_2step(i)`
- `m = lev2knots_doubling(i)`
- `m = lev2knots_tripling(i)`

  $m(i) = 3^{i-1}$

Sparse grids “ingredients”: nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- `m = lev2knots_lin(i)`
- `m = lev2knots_2step(i)`
- `m = lev2knots_doubling(i)`
- `m = lev2knots_tripling(i)`
- `m = lev2knots_GK(i)`

it is possible to specify different **`m`** and **`knots`** in each direction

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

presets for **m,Ifun** are available:

```
[m,Ifun]=define_functions_for_rule(<'TP','TD','HC','SM'>,<N,g>)
```

where for $g \in \mathbb{R}_+^N$, $w \in \mathbb{N}$

- **'TP'** $=$ tensor prod., $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}_+^N : \max_n g_n(i_n - 1) \leq w\right\}$, $m(i) = i$

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

presets for **m,Ifun** are available:

```
[m,Ifun]=define_functions_for_rule(<'TP','TD','HC','SM'>,<N,g>)
```

where for $g \in \mathbb{R}^N_+$, $w \in \mathbb{N}$

- **'TP'** = tensor prod., $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}^N_+ : \max_n g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'TD'** = total deg., $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}^N_+ : \sum_{n=1}^N g_n(i_n - 1) \leq w\right\}$, $m(i) = i$

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

presets for **m,Ifun** are available:

```
[m,Ifun]=define_functions_for_rule(<'TP','TD','HC','SM'>,<N,g>)
```

where for $g \in \mathbb{R}_+^N$, $w \in \mathbb{N}$

- **'TP'** = tensor prod., $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}_+^N : \max_n g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'TD'** = total deg., $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}_+^N : \sum_{n=1}^N g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'HC'** = hyperbolic cross, $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}_+^N : \prod_{n=1}^N i_n^{g_n} \leq w\right\}$, $m(i) = i$

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

presets for **m,Ifun** are available:

```
[m,Ifun]=define_functions_for_rule(<'TP','TD','HC','SM'>,<N,g>)
```

where for $g \in \mathbb{R}_+^N$, $w \in \mathbb{N}$

- **'TP'** = tensor prod., $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}_+^N : \max_n g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'TD'** = total deg., $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}_+^N : \sum_{n=1}^N g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'HC'** = hyperbolic cross, $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}_+^N : \prod_{n=1}^N i_n^{g_n} \leq w\right\}$, $m(i) = i$
- **'SM'** = Smolyak, $\mathcal{I} = \left\{\mathrm{i} \in \mathbb{N}_+^N : \sum_{n=1}^N g_n(i_n - 1) \leq w\right\}$, $m(i) = 2^{i-1} + 1$

Sparse grids "ingredients": nodes, m, $\mathcal{I}$

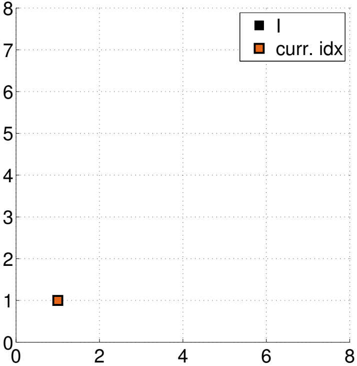
```
N=2;
knots=@(n) knots_CC(n,-1,1);
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

It is also possible to define sparse grids directly by a multi-idx set

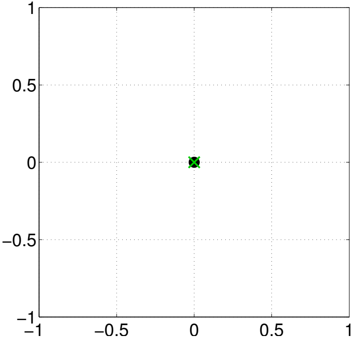
```
% ex. 1) ''hand-typed'' set
C=[1 1; 1 3; 4 1]; % non downward-closed set
[adm,C_compl] = check_set_admissibility(C); % fix C
S_M = smolyak_grid_multiidx_set(C_compl,knots,m);

%ex. 2) create a box in N^2 with top-right corner at [2 3]
jj=[2 3];
D=multiidx_box_set([2 3],1);
T_M = smolyak_grid_multiidx_set(D,knots,m);
```

Sparse grids “ingredients”: adaptive algorithm

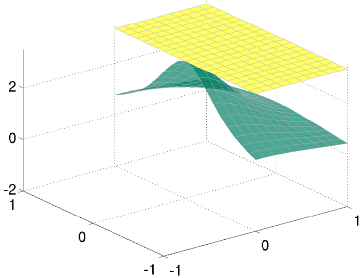
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



Sparse grid set

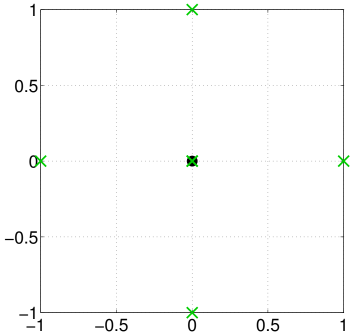


interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

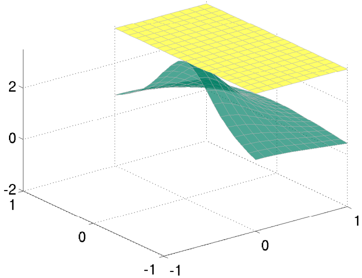
Sparse grids “ingredients”: adaptive algorithm

`res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`

Multi-index set



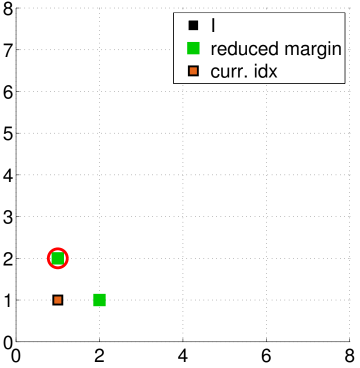
Sparse grid set



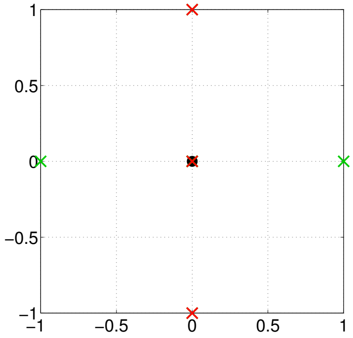
interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids "ingredients": adaptive algorithm

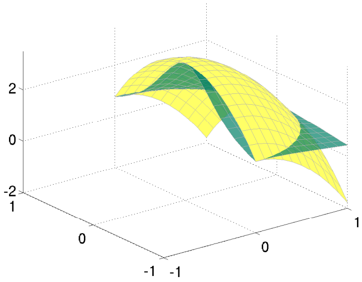
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



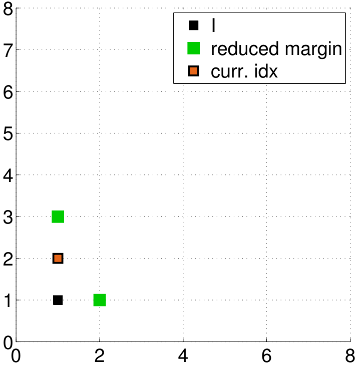
Sparse grid set



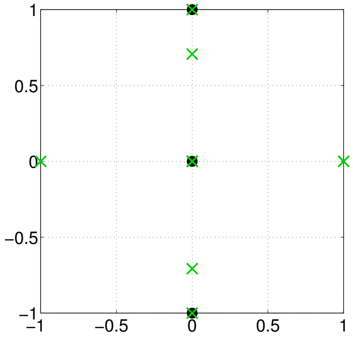
interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

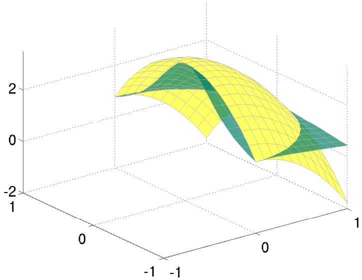
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



Sparse grid set

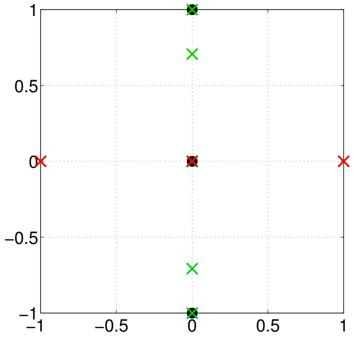


interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

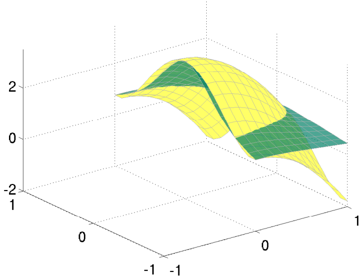
Sparse grids "ingredients": adaptive algorithm

```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```

Multi-index set



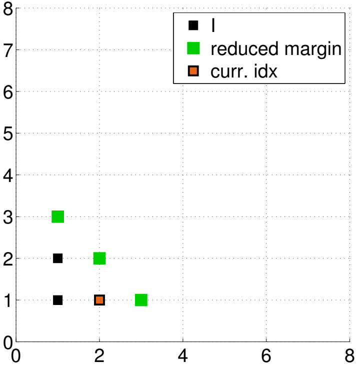
Sparse grid set



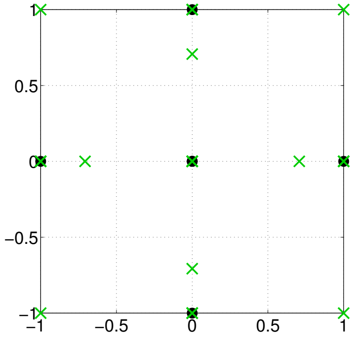
interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids "ingredients": adaptive algorithm

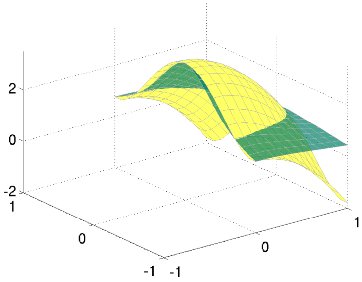
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



Sparse grid set

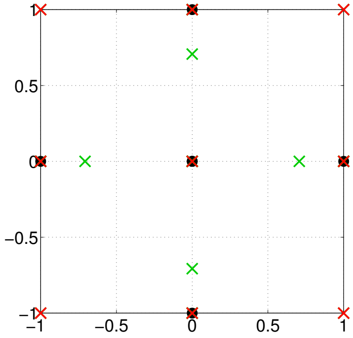


interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

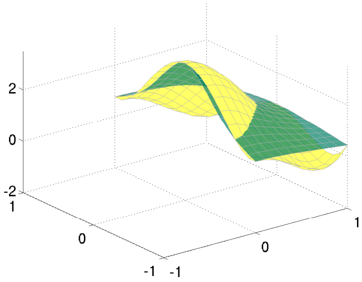
Sparse grids "ingredients": adaptive algorithm

`res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`

Multi-index set



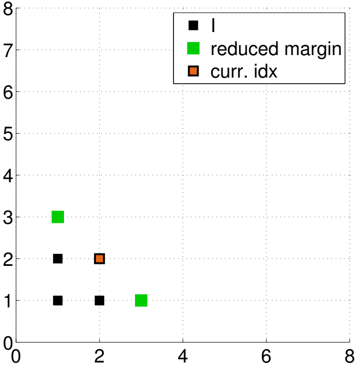
Sparse grid set



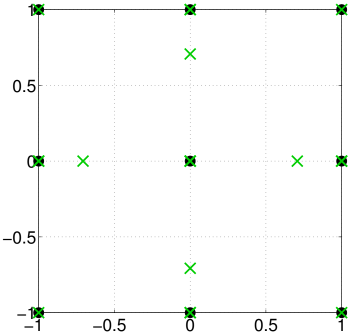
interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

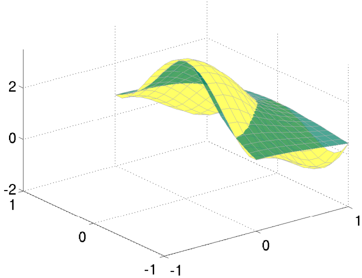
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



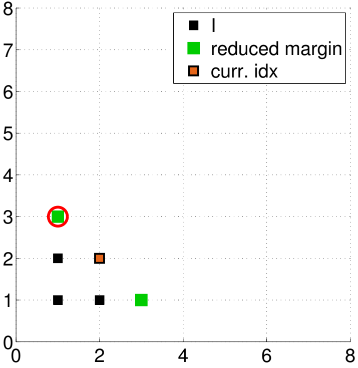
Sparse grid set



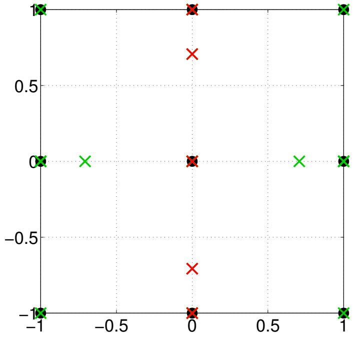
interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

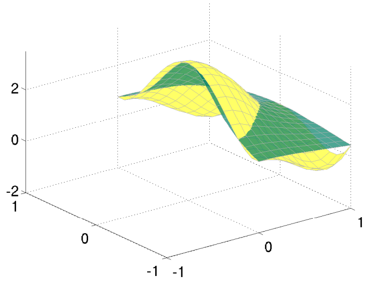
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



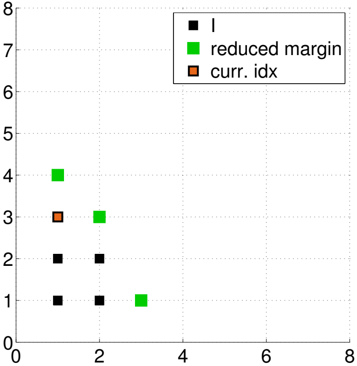
Sparse grid set



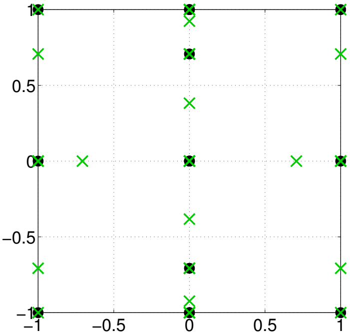
interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids "ingredients": adaptive algorithm

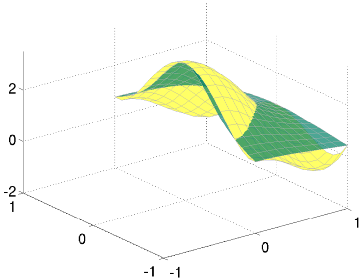
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



Sparse grid set

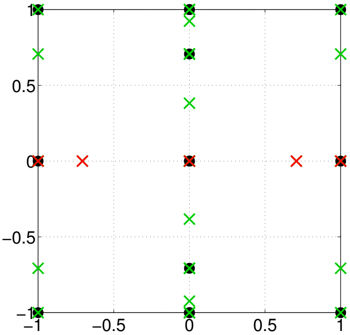


interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

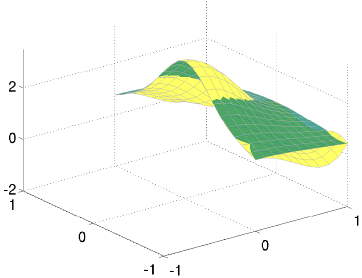
Sparse grids “ingredients”: adaptive algorithm

`res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`

Multi-index set



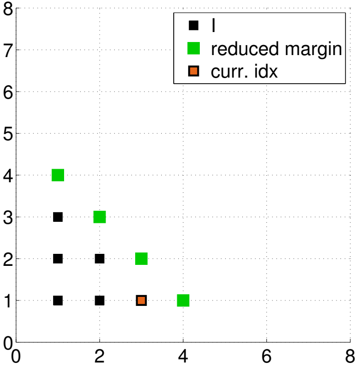
Sparse grid set



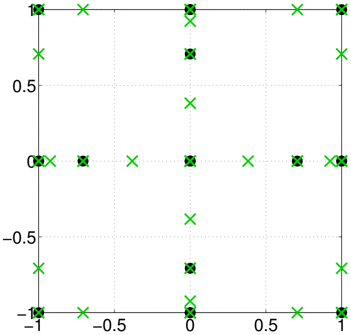
interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

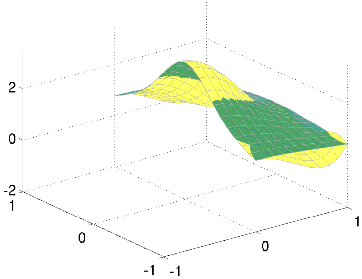
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



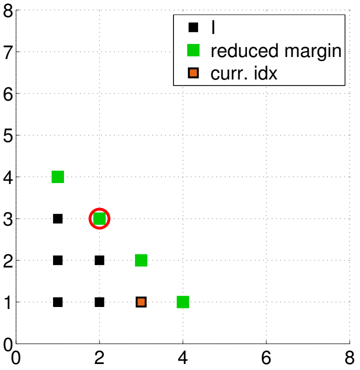
Sparse grid set



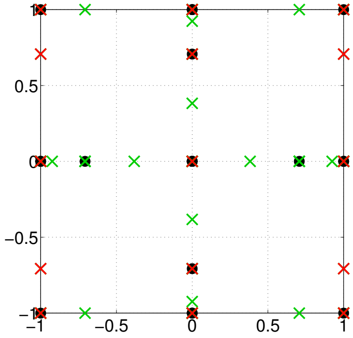
interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

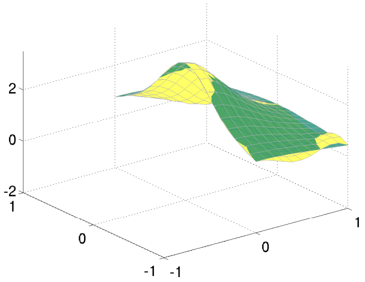
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



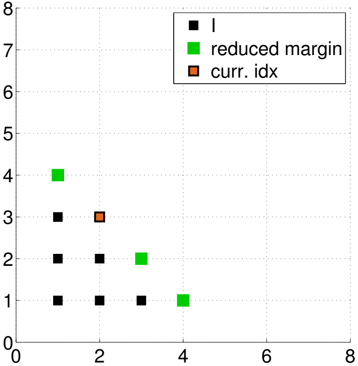
Sparse grid set



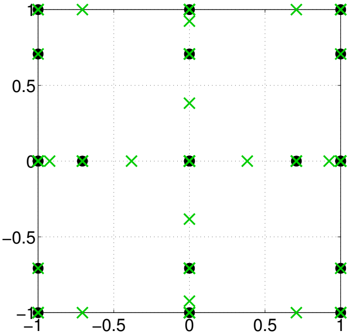
interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

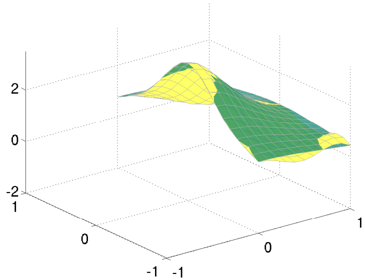
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



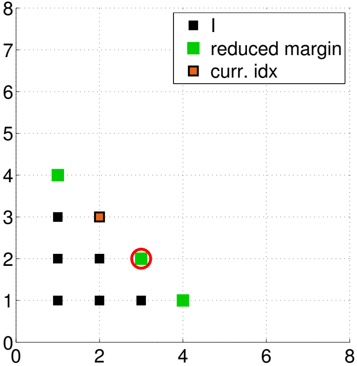
Sparse grid set



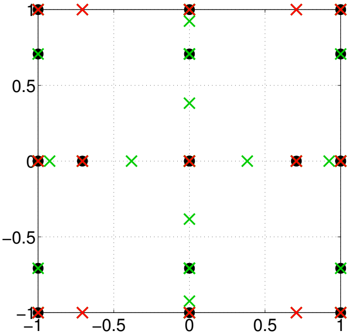
interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids "ingredients": adaptive algorithm

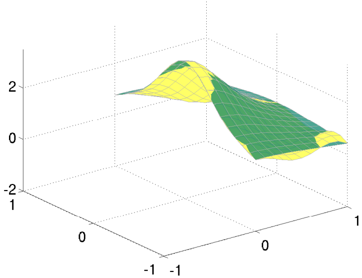
`res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`



Multi-index set



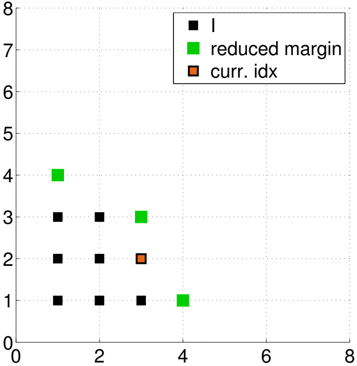
Sparse grid set



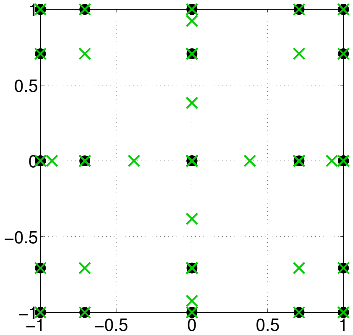
interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

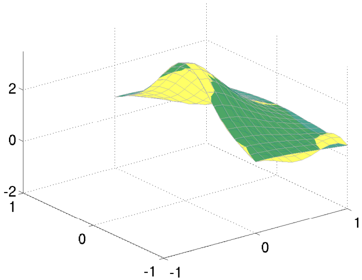
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



Sparse grid set

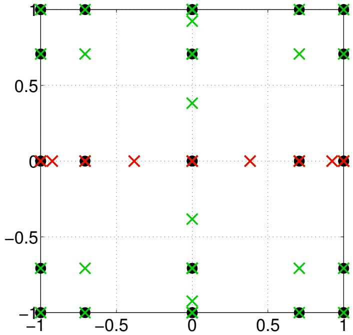


interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

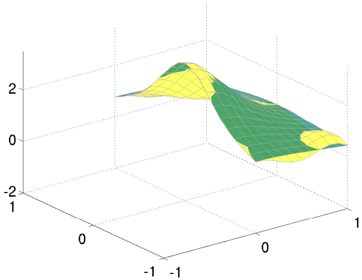
Sparse grids "ingredients": adaptive algorithm

```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```

Multi-index set



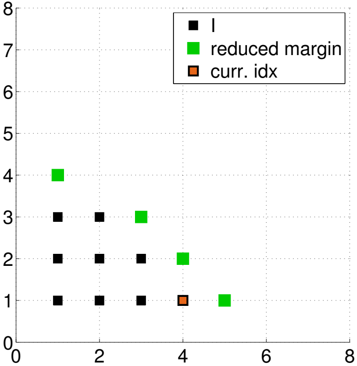
Sparse grid set



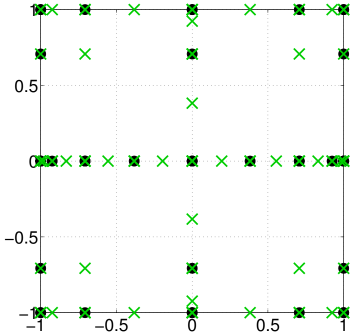
interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

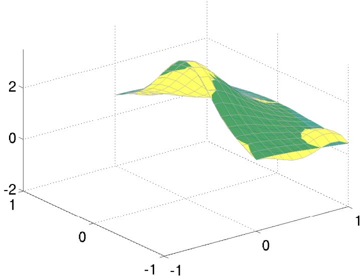
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



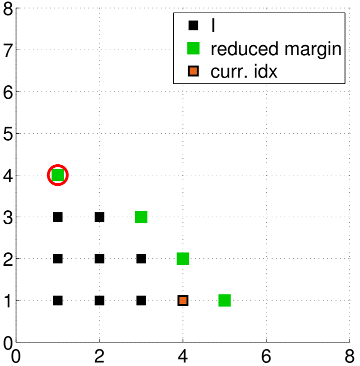
Sparse grid set



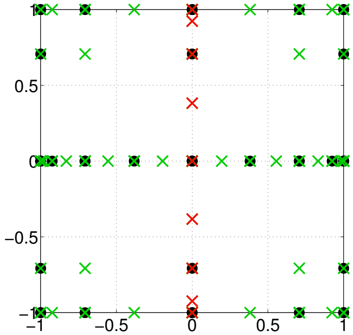
interpolation of $f(\mathbf{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

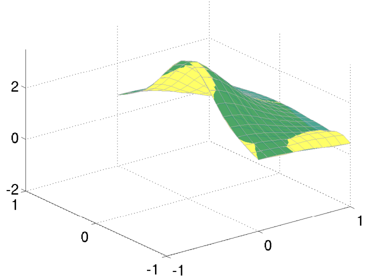
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



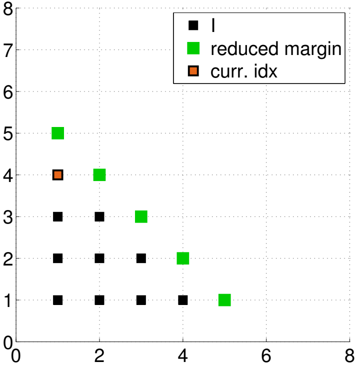
Sparse grid set



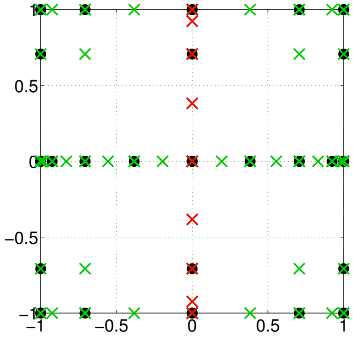
interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Sparse grids “ingredients”: adaptive algorithm

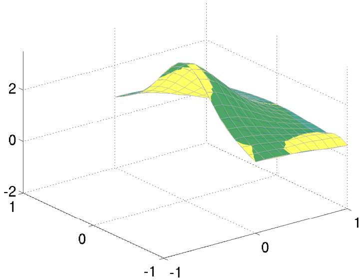
```
res = adapt_sparse_grid(f,N,knots,m,res_old,controls)
```



Multi-index set



Sparse grid set



interpolation of $f(\boldsymbol{y}) = \frac{1}{y_1^2+y_2^2+0.3}$

Outline

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- **`ev_f = evaluate_on_sparse_grid(f,Sr)`**

  can recycle evaluations from previous results if available (regardless of nestedness)

  ```
  ev_f = evaluate_on_sparse_grid(f,S,Sr,ev_f_old,S_old,Sr_old)
  ```

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- **`ev_f = evaluate_on_sparse_grid(f,Sr)`**

  evaluate **`f`** in parallel if more than **`X`** evals are required, uses Matlab parallel toolbox

  ```
  ev_f = evaluate_on_sparse_grid(f,S,Sr,[],[],[],X)
  ```

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- **`ev_f = evaluate_on_sparse_grid(f,Sr)`**
- **`q_f = quadrature_on_sparse_grid(f,Sr)`**

  same features as **`evaluate`**

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`

  `P` is a matrix of eval. points (stored as columns)

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
  - Converts a sparse grid into its equivalent Polynomial Chaos Exp.

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
  - Converts a sparse grid into its equivalent Polynomial Chaos Exp.
  - **Idea:** For each tensor grid in the combination technique, compute the equivalent PCE by solving a Vandermonde system

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
    - Converts a sparse grid into its equivalent Polynomial Chaos Exp.
    - **Idea:** For each tensor grid in the combination technique, compute the equivalent PCE by solving a Vandermonde system
    - Vandermonde matrix is orthogonal for Gaussian quadrature points

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
  - Converts a sparse grid into its equivalent Polynomial Chaos Exp.
  - **Idea:** For each tensor grid in the combination technique, compute the equivalent PCE by solving a Vandermonde system
  - Vandermonde matrix is orthogonal for Gaussian quadrature points
  - several orthogonal polynomials:

    `'Legendre','Hermite','Chebyshev','Laguerre','Jacobi','Generalized Laguerre'`

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`
  - `Si` are the principal Sobol indices of x_i (fraction of variability due to x_i only)

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`
  - `Si` are the principal Sobol indices of x_i (fraction of variability due to x_i only)
  - `Ti` are the total Sobol indices of x_i (fraction of variability due to x_i alone and together with any other variable)

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`
- `grads = derive_sparse_grid(S,Sr,ev_f,P)`

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`
- `grads = derive_sparse_grid(S,Sr,ev_f,P)`

  uses Finite Differences, increment step can be specified

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`
- `grads = derive_sparse_grid(S,Sr,ev_f,P)`
- `H = hessian_sparse_grid(S,Sr,ev_f,P)`

Main functions

```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`
- `grads = derive_sparse_grid(S,Sr,ev_f,P)`
- `H = hessian_sparse_grid(S,Sr,ev_f,P)`

  uses Finite Differences, increment step can be specified

Main functions

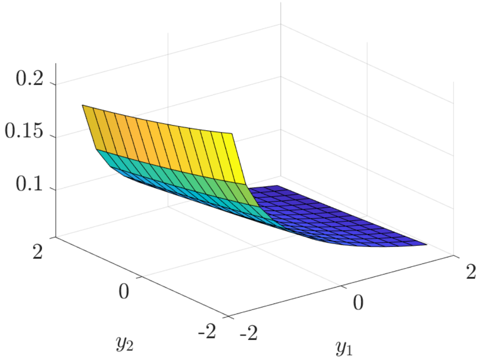
```
f=@(x) x.^2; %vector-valued function
N=2; w=3;
S=smolyak_grid(N,w,@(n) knots_uniform(n,-1,1),@lev2knots_lin);
Sr= reduce_sparse_grid(S);
```

- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`
- `grads = derive_sparse_grid(S,Sr,ev_f,P)`
- `H = hessian_sparse_grid(S,Sr,ev_f,P)`
- plus of course, plotting and exporting on file...

Outline

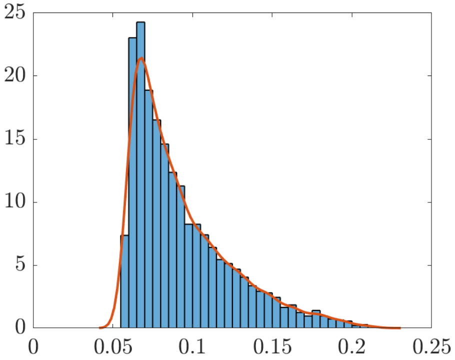
Results with code

```
N = 2;
knots = @(n) knots_CC(n,-sqrt(3),sqrt(3));
lev2knots = @lev2knots_doubling;
rule = @(ii) sum((ii-1));
w=4;
S = smolyak_grid(N,w,knots,lev2knots,rule);
Sr = reduce_sparse_grid(S);

% solve PDEs on sparse grid (expensive part)
I_on_Sr = evaluate_on_sparse_grid(I,Sr);

% task 1: expected value
exp_I = quadrature_on_sparse_grid(I_on_Sr,Sr);

% task 2: evaluate metamodel
yy = linspace(-sqrt(3),sqrt(3),15);
[Y1,Y2] = meshgrid(yy,yy);
eval_points = [Y1(:)';Y2(:)'];
I_vals = interpolate_on_sparse_grid(S,Sr,...
                           I_on_Sr,eval_points);
```



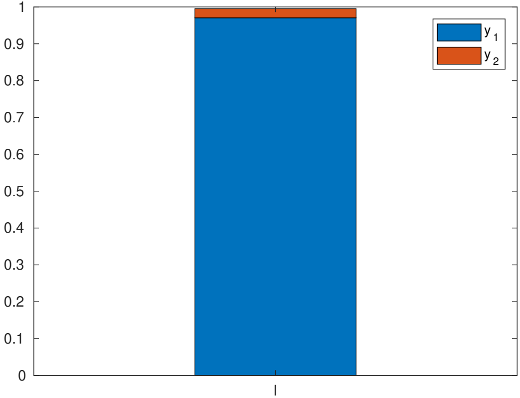
Results with code

```
% task 3: pdf
M = 5000;
y_samples = -sqrt(3)+2*sqrt(3)*rand(N,M);
I_vals_rand = interpolate_on_sparse_grid(S,Sr,...
                                I_on_Sr,y_samples);
H = histogram(I_vals_rand,'Normalization','pdf');
[pdf_vals,pdf_pts] = ksdensity(I_vals_rand);
```



Results with code

```
% task 4: sensitivity analysis (Sobol indices)
domain = [-sqrt(3) -sqrt(3);sqrt(3) sqrt(3)];
Sob = compute_sobol_indices_from_sparse_ grid(...
          S,Sr,I on Sr ,domain,'legendre');
bar(1,Sob','stacked')
```



Outline

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- high-dimensional quadrature/interpolation, geared towards UQ

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- high-dimensional quadrature/interpolation, geared towards UQ
- next release (2023-??): Multi-Index Stochastic Collocation. Uses multiple fidelities for PDEs to reduce computational costs

Bibliography

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- high-dimensional quadrature/interpolation, geared towards UQ
- next release (2023-??): Multi-Index Stochastic Collocation. Uses multiple fidelities for PDEs to reduce computational costs

Bibliography

- C. Piazzola, L. Tamellini, *The Sparse Grids Matlab Kit - a Matlab implementation of sparse grids for high-dimensional function approximation and uncertainty quantification*, 2022.

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- high-dimensional quadrature/interpolation, geared towards UQ
- next release (2023-??): Multi-Index Stochastic Collocation. Uses multiple fidelities for PDEs to reduce computational costs

Bibliography

- C. Piazzola, L. Tamellini, *The Sparse Grids Matlab Kit - a Matlab implementation of sparse grids for high-dimensional function approximation and uncertainty quantification*, 2022.
- `https://sites.google.com/view/sparse-grids-kit`

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- high-dimensional quadrature/interpolation, geared towards UQ
- next release (2023-??): Multi-Index Stochastic Collocation. Uses multiple fidelities for PDEs to reduce computational costs

Bibliography

- C. Piazzola, L. Tamellini, *The Sparse Grids Matlab Kit - a Matlab implementation of sparse grids for high-dimensional function approximation and uncertainty quantification*, 2022.
- `https://sites.google.com/view/sparse-grids-kit`
- J. Martínez-Frutos, F. Periago Esparza, *Optimal Control of PDEs under Uncertainty,* Springer 2018. Uses Matlab Sparse Grids Kit