Sparse grids Matlab kit

L.Tamellini[1], F. Nobile[2]

[1] CNR-IMATI, Pavia
[2] CSQI-MATHICSE, EPFL, Switzerland

October 15, 2018

Outline

1. Basic data structure
2. Main features
3. Numerical examples
4. Conclusions

Contributors, releases

- main contributors: Lorenzo Tamellini, Fabio Nobile

Contributors, releases

- main contributors: Lorenzo Tamellini, Fabio Nobile
- other contributors: Diane Guignard, Giovanni Porta, Björn Sprungk, Francesco Tesei

Contributors, releases

- main contributors: Lorenzo Tamellini, Fabio Nobile
- other contributors: Diane Guignard, Giovanni Porta, Björn Sprungk, Francesco Tesei
- roughly yearly releases:

Contributors, releases

- main contributors: Lorenzo Tamellini, Fabio Nobile
- other contributors: Diane Guignard, Giovanni Porta, Björn Sprungk, Francesco Tesei
- roughly yearly releases:
  - 18-10 ("Esperanza")
  - 17-5 ("Trent")
  - 15-8 ("Woodstock")
  - 14-12 ("Fenice")
  - 14-4 ("Ritchie")

Contributors, releases

- main contributors: Lorenzo Tamellini, Fabio Nobile
- other contributors: Diane Guignard, Giovanni Porta, Björn Sprungk, Francesco Tesei
- roughly yearly releases:
  - 18-10 (“Esperanza”)
  - 17-5 (“Trent”)
  - 15-8 (“Woodstock”)
  - 14-12 (“Fenice”)
  - 14-4 (“Ritchie”)
- Download from `https://csqi.epfl.ch`

Contributors, releases

- main contributors: Lorenzo Tamellini, Fabio Nobile
- other contributors: Diane Guignard, Giovanni Porta, Björn Sprungk, Francesco Tesei
- roughly yearly releases:
  - 18-10 (“Esperanza”)
  - 17-5 (“Trent”)
  - 15-8 (“Woodstock”)
  - 14-12 (“Fenice”)
  - 14-4 (“Ritchie”)
- Download from `https://csqi.epfl.ch`
- 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 (7000 lines of code, 3600 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 (7000 lines of code, 3600 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 (7000 lines of code, 3600 lines of comments)
- Geared towards UQ, but flexible enough for other purposes

Aim of these slides: give rough idea of structure, show by examples features and ease of use

Outline

1 Basic data structure

2 Main features

3 Numerical examples

4 Conclusions

Backbone: combination technique & “reduced” sparse grid

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

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

Backbone: combination technique & “reduced” sparse grid

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

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

Backbone: combination technique & “reduced” sparse grid

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

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

Backbone: combination technique & “reduced” sparse grid

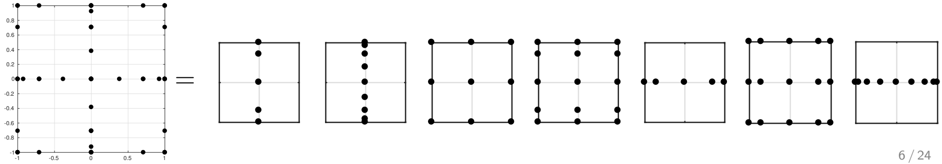
```
N=2;
knots=@(n) knots_CC(n,-1,1,'nonprob');
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

```
>> 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,'nonprob');
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,'nonprob');
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,'nonprob');
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,'nonprob');
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,'nonprob');
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,'nonprob');
w = 3;
m = @lev2knots_doubling;
Ifun = @(i) sum(i-1);
S = smolyak_grid(N,w,knots,m,Ifun);
```

- `[x,w]=knots_CC(n,a,b) %Clenshaw-Curtis`

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

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

- `[x,w]=knots_CC(n,a,b) %Clenshaw-Curtis`
- `[x,w]=knots_uniform(n,a,b) %Gauss-Legendre`

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

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

- `[x,w]=knots_CC(n,a,b) %Clenshaw-Curtis`
- `[x,w]=knots_uniform(n,a,b) %Gauss-Legendre`
- `[x,w]=knots_leja(n,a,b,'line')`

  $x_1 = b$

  $x_2 = a$

  $x_3 = (a+b)/2$

  $x_n = \mathrm{argmax}_{[a\ b]} \prod_{k=1}^{n-1} (x - x_k)$

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

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

- `[x,w]=knots_CC(n,a,b) %Clenshaw-Curtis`
- `[x,w]=knots_uniform(n,a,b) %Gauss-Legendre`
- `[x,w]=knots_leja(n,a,b,'line')`
- `[x,w]=knots_leja(n,a,b,'sym_line')`

$x_1 = b$
$x_2 = a$
$x_3 = (a+b)/2$
$x_{2n} = \text{argmax}_{[a\ b]} \prod_{k=1}^{2n-1} (x - x_k)$
$x_{2n+1} = \text{symmetric of } x_{2n} \text{ wrt } (a+b)/2$

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

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

- `[x,w]=knots_CC(n,a,b) %Clenshaw-Curtis`
- `[x,w]=knots_uniform(n,a,b) %Gauss-Legendre`
- `[x,w]=knots_leja(n,a,b,'line')`
- `[x,w]=knots_leja(n,a,b,'sym_line')`
- `[x,w]=knots_leja(n,a,b,'p_disk')`

  compute Leja pts on the complex unit ball, and project on the real line

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

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

- `[x,w]=knots_CC(n,a,b) %Clenshaw-Curtis`
- `[x,w]=knots_uniform(n,a,b) %Gauss-Legendre`
- `[x,w]=knots_leja(n,a,b,'line')`
- `[x,w]=knots_leja(n,a,b,'sym_line')`
- `[x,w]=knots_leja(n,a,b,'p_disk')`
- `[x,w]=knots_gaussian(n,mi,sigma) %Gauss-Hermite`

  for gaussian weights with mean μ and st. dev. σ

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

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

- `[x,w]=knots_CC(n,a,b) %Clenshaw-Curtis`
- `[x,w]=knots_uniform(n,a,b) %Gauss-Legendre`
- `[x,w]=knots_leja(n,a,b,'line')`
- `[x,w]=knots_leja(n,a,b,'sym_line')`
- `[x,w]=knots_leja(n,a,b,'p_disk')`
- `[x,w]=knots_gaussian(n,mi,sigma) %Gauss-Hermite`
- `[x,w]=knots_kpn(n)%Kronrod Patteron Normal, Genz-Keister`

  Tabulated sequence of nested extensions of (n+1) Gauss–Hermite with maximal exactness degree: $m = 1, 3, 9, 19, 35$

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

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

- `[x,w]=knots_CC(n,a,b) %Clenshaw-Curtis`
- `[x,w]=knots_uniform(n,a,b) %Gauss-Legendre`
- `[x,w]=knots_leja(n,a,b,'line')`
- `[x,w]=knots_leja(n,a,b,'sym_line')`
- `[x,w]=knots_leja(n,a,b,'p_disk')`
- `[x,w]=knots_gaussian(n,mi,sigma) %Gauss-Hermite`
- `[x,w]=knots_kpn(n)%Kronrod Patteron Normal, Genz-Keister`
- `[x,w]=knots_gaussian_leja(n)%Narayan-Jakeman, SISC 2014`

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

```
N=2;
knots=@(n) knots_CC(n,-1,1,'nonprob');
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,'nonprob');
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,'nonprob');
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,'nonprob');
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_kpn(i)`

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

```
N=2;
knots=@(n) knots_CC(n,-1,1,'nonprob');
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_kpn(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,'nonprob');
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., $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}_+^N : \max_n g_n(i_n - 1) \leq w\right\}$, $m(i) = i$

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

```
N=2;
knots=@(n) knots_CC(n,-1,1,'nonprob');
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., $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}_+^N : \max_n g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'TD'** = total deg., $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}_+^N : \sum_{n=1}^N g_n(i_n - 1) \leq w\right\}$, $m(i) = i$

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

```
N=2;
knots=@(n) knots_CC(n,-1,1,'nonprob');
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., $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}_+^N : \max_n g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'TD'** $=$ total deg., $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}_+^N : \sum_{n=1}^N g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'HC'** $=$ hyperbolic cross, $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}_+^N : \prod_{n=1}^N i_n^{g_n} \leq w\right\}$, $m(i) = i$

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

```
N=2;
knots=@(n) knots_CC(n,-1,1,'nonprob');
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., $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}^N_+ : \max_n g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'TD'** $=$ total deg., $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}^N_+ : \sum_{n=1}^N g_n(i_n - 1) \leq w\right\}$, $m(i) = i$
- **'HC'** $=$ hyperbolic cross, $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}^N_+ : \prod_{n=1}^N i_n^{g_n} \leq w\right\}$, $m(i) = i$
- **'SM'** $=$ Smolyak, $\mathcal{I} = \left\{\mathbf{i} \in \mathbb{N}^N_+ : \sum_{n=1}^N g_n(i_n - 1) \leq w\right\}$, $m(i) = 2^{i-1} + 1$

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

```
N=2;
knots=@(n) knots_CC(n,-1,1,'nonprob');
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);
```

Outline

1. Basic data structure
2. Main features
3. Numerical examples
4. Conclusions

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

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - `knots` can be non-nested and on unbounded interval

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - **`knots`** can be non-nested and on unbounded interval
  - comes with several definitons of profit/surplus

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - `knots` can be non-nested and on unbounded interval
  - comes with several definitons of profit/surplus
    1. difference between consecutive quadratures

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - `knots` can be non-nested and on unbounded interval
  - comes with several definitons of profit/surplus
    1. difference between consecutive quadratures
    2. max of difference between two consecutive sparse grid approx

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - **knots** can be non-nested and on unbounded interval
  - comes with several definitons of profit/surplus
    1. difference between consecutive quadratures
    2. max of difference between two consecutive sparse grid approx
       - for nested points: identical to max between sparse grid and true fun.
       - works for non-nested points too
       - over the last added tensor grid

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - **knots** can be non-nested and on unbounded interval
  - comes with several definitons of profit/surplus
    1. difference between consecutive quadratures
    2. max of difference between two consecutive sparse grid approx
    3. weighting by nb of points and arbitary densities is possible

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - **knots** can be non-nested and on unbounded interval
  - comes with several definitons of profit/surplus
    1. difference between consecutive quadratures
    2. max of difference between two consecutive sparse grid approx
    3. weighting by nb of points and arbitary densities is possible
  - computation can be stopped, dumped on variables and restarted

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - `knots` can be non-nested and on unbounded interval
  - comes with several definitons of profit/surplus
    1. difference between consecutive quadratures
    2. max of difference between two consecutive sparse grid approx
    3. weighting by nb of points and arbitary densities is possible
  - computation can be stopped, dumped on variables and restarted
  - a buffer of N_b “explored but unused variables” can be set

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
    - `knots` can be non-nested and on unbounded interval
    - comes with several definitons of profit/surplus
        1. difference between consecutive quadratures
        2. max of difference between two consecutive sparse grid approx
        3. weighting by nb of points and arbitary densities is possible
    - computation can be stopped, dumped on variables and restarted
    - a buffer of N_b "explored but unused variables" can be set

      the algorithm starts with $N_{curr} = N_b$ dim.; as soon as points are placed in one dim., a new one is taken into account, i.e., $N_{curr} = N_{curr} + 1$.
      In this way, there are always N_b dim. whose "initial profit" is computed but along which no point is placed

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
  - `knots` can be non-nested and on unbounded interval
  - comes with several definitons of profit/surplus
    1. difference between consecutive quadratures
    2. max of difference between two consecutive sparse grid approx
    3. weighting by nb of points and arbitary densities is possible
  - computation can be stopped, dumped on variables and restarted
  - a buffer of N_b “explored but unused variables” can be set
  - support vector-valued functions, appropriate profits can be set

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[coeffs,I] = convert_to_modal(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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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'`

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `export_sparse_grid_to_file(Sr,'filename')`

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `export_sparse_grid_to_file(Sr,'filename')`

  save points (and optionally quad weights) on an ascii file, one point per row

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `export_sparse_grid_to_file(Sr,'filename')`
- `plot_sparse_grids_interpolant(S,Sr,f_ev)`

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `export_sparse_grid_to_file(Sr,'filename')`
- `plot_sparse_grids_interpolant(S,Sr,f_ev)`

  plots the sparse grids interpolant of `f`. Different plots are produced for the cases $N = 2, N = 3, N > 3$ (see next slide)

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)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `export_sparse_grid_to_file(Sr,'filename')`
- `plot_sparse_grids_interpolant(S,Sr,f_ev)`
- `plot_sparse_grids(S)` or `plots_sparse_grids(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)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[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)`
- `export_sparse_grid_to_file(Sr,'filename')`
- `plot_sparse_grids_interpolant(S,Sr,f_ev)`
- `plot_sparse_grids(S)` or `plots_sparse_grids(Sr)`

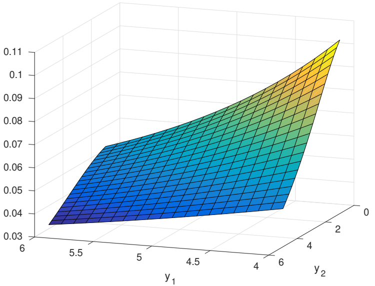
  plots (selected coordinates of) the sparse grid points in a 2d plane

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);
```

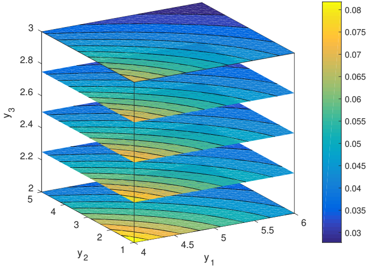
- `ev_f = evaluate_on_sparse_grid(f,Sr)`
- `q_f = quadrature_on_sparse_grid(f,Sr)`
- `int_f = interpolate_on_sparse_grid(S,Sr,ev_f,P)`
- `res = adapt_sparse_grid(f,N,knots,m,res_old,controls)`
- `[coeffs,I] = convert_to_modal(S,Sr,ev_f,'Legendre')`
- `[Si,Ti]=compute_sobol_indices_from_sparse_grid(S,Sr,ev_f,'Legendre')`
- `grads = derive_sparse_grid(S,Sr,ev_f,P)`
- `export_sparse_grid_to_file(Sr,'filename')`
- `plot_sparse_grids_interpolant(S,Sr,f_ev)`
- `plot_sparse_grids(S)` or `plots_sparse_grids(Sr)`

plot_sparse_grids_interpolant - examples



case N=2: surf of sparse grid interpolant.

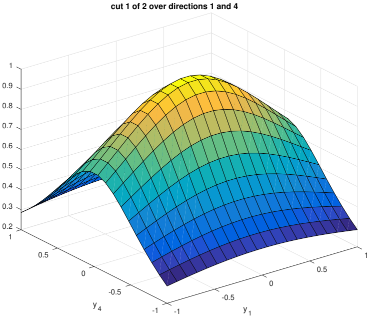
Above, $f(x) = 1/(1 + 0.5x_1^2 + 0.5x_2^2)$

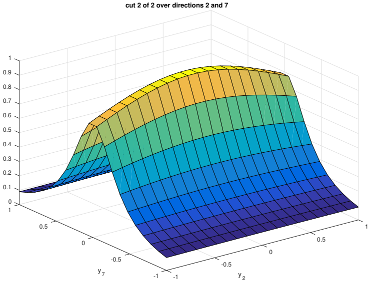


case N=3: several contourf plots of the sparse grid interpolant obtained fixing x_3 at different values will be plotted in the same axes.

Above, $f(x) = 1/(1 + 0.5x_1^2 + 0.5x_2^2 + 0.5x_3^2)$

plot_sparse_grids_interpolant - examples





case N>3: several two-dimensional surf plots will be produced. In each of them, all variables are frozen to their average value but 2, and the value of the interpolant with respect of those will be plotted.

Above, $f(x) = 1./(1 + 0.5x_1^2 + 0.25x_2^2 + 5x_3^2 + 2x_4^2 + 0.001x_5^2 + 10x_6^2 + 10x_7^2)$. We plot the two-dimesional cuts (x_1, x_4) and (x_2, x_7)

Outline

1. Basic data structure
2. Main features
3. Numerical examples
4. Conclusions

Examples of applications

1. Convergence of sparse grids approximation of lognormal problem:
    - **adapt_sparse_grid** with non-nested knots on unbounded domains
    - **smolyak_grid_multiidx_set**
    - **convert_to_modal**

Examples of applications

1. Convergence of sparse grids approximation of lognormal problem:
   - **`adapt_sparse_grid`** with non-nested knots on unbounded domains
   - **`smolyak_grid_multiidx_set`**
   - **`convert_to_modal`**

2. Geochemical compaction
   - very easy connection with built-in Matlab routines (in this case, **`fminsearch`**)
   - **`derive_sparse_grid`**

Ex. 1) Elliptic PDE with lognormal diffusion coefficient

$$\begin{cases} -\dfrac{\mathrm{d}}{\mathrm{d}x}\left(a(x,\boldsymbol{\xi})\,\dfrac{\mathrm{d}}{\mathrm{d}x}u(x,\boldsymbol{\xi})\right) = 0.03\sin(2\pi x), \qquad u(0,\boldsymbol{\xi}) = u(1,\boldsymbol{\xi}) = 0 \\ \log a(x,\boldsymbol{\xi}) = 0.1\displaystyle\sum_{m=1}^{\infty} \underbrace{\frac{\sqrt{2}}{(\pi m)^{q}}\sin(m\pi x)}_{=:\phi_m(x)}\,\xi_m, \qquad q \geq 1,\ \text{smoothed Brownian bridge} \end{cases}$$

Ex. 1) Elliptic PDE with lognormal diffusion coefficient

$$
\begin{cases}
-\dfrac{\mathrm{d}}{\mathrm{d}x}\left(a(x,\boldsymbol{\xi})\,\dfrac{\mathrm{d}}{\mathrm{d}x}u(x,\boldsymbol{\xi})\right) = 0.03\sin(2\pi x), \qquad u(0,\boldsymbol{\xi}) = u(1,\boldsymbol{\xi}) = 0 \\
\log a(x,\boldsymbol{\xi}) = 0.1\displaystyle\sum_{m=1}^{\infty} \underbrace{\frac{\sqrt{2}}{(\pi m)^{q}}\sin(m\pi x)}_{=:\phi_m(x)}\,\xi_m, \qquad q \geq 1,\ \text{smoothed Brownian bridge}
\end{cases}
$$

We have a semi-analytic formula to compute $u(\cdot,\boldsymbol{\xi})$

Ex. 1) Elliptic PDE with lognormal diffusion coefficient

$$
\begin{cases}
-\dfrac{\mathrm{d}}{\mathrm{d}x}\left(a(x,\boldsymbol{\xi})\,\dfrac{\mathrm{d}}{\mathrm{d}x}u(x,\boldsymbol{\xi})\right) = 0.03\sin(2\pi x), \qquad u(0,\boldsymbol{\xi}) = u(1,\boldsymbol{\xi}) = 0 \\
\log a(x,\boldsymbol{\xi}) = 0.1\displaystyle\sum_{m=1}^{\infty} \underbrace{\frac{\sqrt{2}}{(\pi m)^q}\sin(m\pi x)}_{=:\phi_m(x)}\,\xi_m, \qquad q \geq 1,\ \text{smoothed Brownian bridge}
\end{cases}
$$

We have a semi-analytic formula to compute $u(\cdot,\boldsymbol{\xi})$

Convergence Theorem

- Under assumptions on $a(x,\boldsymbol{\xi})$ and knots, $\exists$ a multiidx set Λ_N with $|\Lambda_N| = N$ st

$$
\|u - U_{\Lambda_N}u\|_{L^2_\mu} \;\leq\; CN^{-(q-1.5)} \;\leq\; C|\Xi_{\Lambda_N}|^{-\left(\frac{q-1.5}{2}\right)}.
$$

Ex. 1) Elliptic PDE with lognormal diffusion coefficient

$$
\begin{cases}
-\dfrac{\mathrm{d}}{\mathrm{d}x}\left(a(x,\boldsymbol{\xi})\,\dfrac{\mathrm{d}}{\mathrm{d}x}u(x,\boldsymbol{\xi})\right) = 0.03\sin(2\pi x), \qquad u(0,\boldsymbol{\xi}) = u(1,\boldsymbol{\xi}) = 0 \\
\log a(x,\boldsymbol{\xi}) = 0.1\sum_{m=1}^{\infty} \underbrace{\dfrac{\sqrt{2}}{(\pi m)^q}\sin(m\pi x)}_{=:\phi_m(x)}\,\xi_m, \qquad q \geq 1, \text{ smoothed Brownian bridge}
\end{cases}
$$

We have a semi-analytic formula to compute $u(\cdot,\boldsymbol{\xi})$

Convergence Theorem

- Under assumptions on $a(x,\boldsymbol{\xi})$ and knots, $\exists$ a multiidx set Λ_N with $|\Lambda_N| = N$ st

$$\|u - U_{\Lambda_N}u\|_{L^2_\mu} \leq CN^{-(q-1.5)} \leq C|\Xi_{\Lambda_N}|^{-\left(\frac{q-1.5}{2}\right)}.$$

- constructive, provides an estimate of the optimal set Λ_N:

$$\Lambda_N = \{\boldsymbol{\nu} \in \mathbb{N}^{\mathbb{N}} \text{ with N largest } \hat{c}_{\boldsymbol{\nu}}\}, \qquad \hat{c}_{\boldsymbol{\nu}} = \hat{c}_{\boldsymbol{\nu}}(q)$$

Ex. 1) Elliptic PDE with lognormal diffusion coefficient

$$\begin{cases} -\dfrac{\mathrm{d}}{\mathrm{d}x}\left(a(x,\boldsymbol{\xi})\,\dfrac{\mathrm{d}}{\mathrm{d}x}u(x,\boldsymbol{\xi})\right) = 0.03\sin(2\pi x), \qquad u(0,\boldsymbol{\xi}) = u(1,\boldsymbol{\xi}) = 0 \\ \log a(x,\boldsymbol{\xi}) = 0.1\displaystyle\sum_{m=1}^{\infty} \underbrace{\frac{\sqrt{2}}{(\pi m)^q}\sin(m\pi x)}_{=:\phi_m(x)}\,\xi_m, \qquad q \geq 1,\ \text{smoothed Brownian bridge} \end{cases}$$

We have a semi-analytic formula to compute $u(\cdot,\boldsymbol{\xi})$

Convergence Theorem

- Under assumptions on $a(x,\boldsymbol{\xi})$ and knots, $\exists$ a multiidx set Λ_N with $|\Lambda_N| = N$ st

$$\|u - U_{\Lambda_N} u\|_{L^2_\mu} \leq CN^{-(q-1.5)} \leq C|\Xi_{\Lambda_N}|^{-\left(\frac{q-1.5}{2}\right)}.$$

- constructive, provides an estimate of the optimal set Λ_N:

$$\Lambda_N = \{\boldsymbol{\nu} \in \mathbb{N}^{\mathbb{N}} \text{ with N largest } \hat{c}_{\boldsymbol{\nu}}\}, \qquad \hat{c}_{\boldsymbol{\nu}} = \hat{c}_{\boldsymbol{\nu}}(q)$$

- technical assumptions on knots proved for Gauss–Hermite knots (so far).

Computational details

- In practice, we build the **a-priori set** by running the usual dimension-adaptive algorithm with profit $= \hat{c}_{\boldsymbol{\nu}}$ from the constructive proof

Computational details

- In practice, we build the **a-priori set** by running the usual dimension-adaptive algorithm with profit $= \hat{c}_{\boldsymbol{\nu}}$ from the constructive proof
- Compare against classical **a-posteriori:** $\|\Delta_{\boldsymbol{\nu}} u\|_{L^2_\mu(\Gamma;H^1_0([0,1]))} / |\Xi^{(\boldsymbol{\nu})}|$

Computational details

- In practice, we build the **a-priori set** by running the usual dimension-adaptive algorithm with profit $= \hat{c}_{\boldsymbol{\nu}}$ from the constructive proof
- Compare against classical **a-posteriori:** $\|\Delta_{\boldsymbol{\nu}} u\|_{L^2_\mu(\Gamma; H^1_0([0,1]))} / |\Xi^{(\boldsymbol{\nu})}|$
- profit computed by quadrature over last tensor grid added, $H^1([0,1])$ norm needed!

Computational details

- In practice, we build the **a-priori set** by running the usual dimension-adaptive algorithm with profit $= \hat{c}_{\boldsymbol{\nu}}$ from the constructive proof
- Compare against classical **a-posteriori:** $\|\Delta_{\boldsymbol{\nu}} u\|_{L^2_\mu(\Gamma;H^1_0([0,1]))} / |\Xi^{(\boldsymbol{\nu})}|$
- profit computed by quadrature over last tensor grid added, $H^1([0,1])$ norm needed!
- choice of N_b is relevant

Computational details

- In practice, we build the **a-priori set** by running the usual dimension-adaptive algorithm with profit $= \hat{c}_{\boldsymbol{\nu}}$ from the constructive proof
- Compare against classical **a-posteriori:** $\|\Delta_{\boldsymbol{\nu}} u\|_{L^2_\mu(\Gamma;H^1_0([0,1]))} / |\Xi^{(\boldsymbol{\nu})}|$
- profit computed by quadrature over last tensor grid added, $H^1([0,1])$ norm needed!
- choice of N_b is relevant
- L^2_μ error norm computed by MC

Computational details

- In practice, we build the **a-priori set** by running the usual dimension-adaptive algorithm with profit $= \hat{c}_{\boldsymbol{\nu}}$ from the constructive proof
- Compare against classical **a-posteriori:** $\|\Delta_{\boldsymbol{\nu}} u\|_{L^2_\mu(\Gamma; H^1_0([0,1]))} / |\Xi^{(\boldsymbol{\nu})}|$
- profit computed by quadrature over last tensor grid added, $H^1([0,1])$ norm needed!
- choice of N_b is relevant
- L^2_μ error norm computed by MC

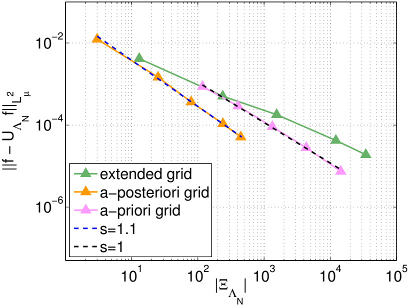
```
% code snippet, a-priori heuristic
Lambda = buildLambda_prior(..);
for iter = 1:P
    C = Lambda(1:P,:);
    C = sortrows(C);
    S = smolyak_grid_multiidx_set(C,..);
    Sr = reduce_sparse_grid(S);
    f_grid=evaluate_on_sparse_grid(f,S,Sr,evals_old,..);
    evals_old = f_grid;
    err = ... % calls interpolate_on_sparse_grid over an MC sample
 end
```

Computational details

- In practice, we build the **a-priori set** by running the usual dimension-adaptive algorithm with profit $= \hat{c}_{\boldsymbol{\nu}}$ from the constructive proof
- Compare against classical **a-posteriori:** $\|\Delta_{\boldsymbol{\nu}} u\|_{L^2_\mu(\Gamma; H^1_0([0,1])}/|\Xi^{(\boldsymbol{\nu})}|$
- profit computed by quadrature over last tensor grid added, $H^1([0,1])$ norm needed!
- choice of N_b is relevant
- L^2_μ error norm computed by MC

```
% code snippet, a-posteriori heuristic
f = @(y) sol(a_coef(y));
controls.op_vect = @(A,B) H1normALL(A-B); % profit for functions
controls.var_buffer_size = 5;
for nb_pts = [10 100 1000..]
    controls.max_pts = nb_pts;
    adapt = adapt_sparse_grid(f,dim,..,prev_adapt,controls);
    prev_adapt = adapt;
    err = .. % calls interpolate_on_sparse_grid over an MC sample
end
```

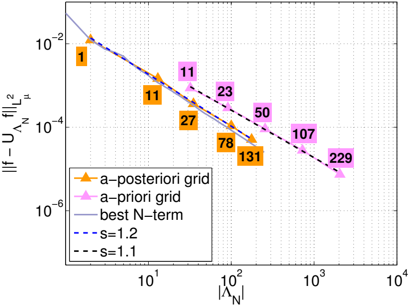
convergence wrt $|\Xi_{\Lambda_N}|$



$q = 2$
expect $s = 0.25$;
a-priori $s = 1.0$;
a-posteriori $s = 1.1$

- Extended grid = a-posteriori with evaluations in the neighbourhood
- Expected rate smaller than observed:
  - summability argument could be improved
  - bound between number of elements and points not sharp

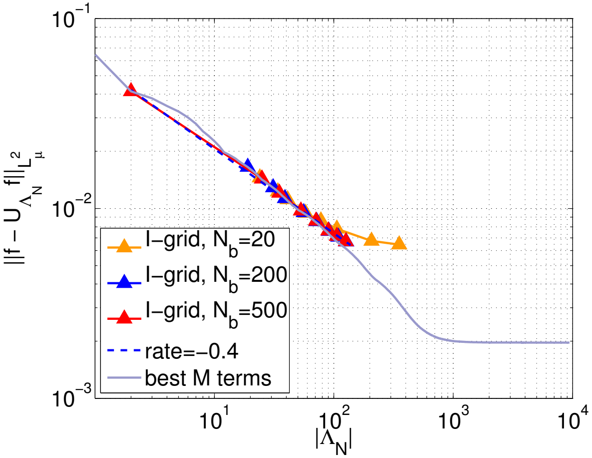
convergence wrt $|\Lambda_N|$



$q = 2$
expect $s = 0.5$;
a-priori $s = 1.1$;
a-posteriori $s = 1.2$

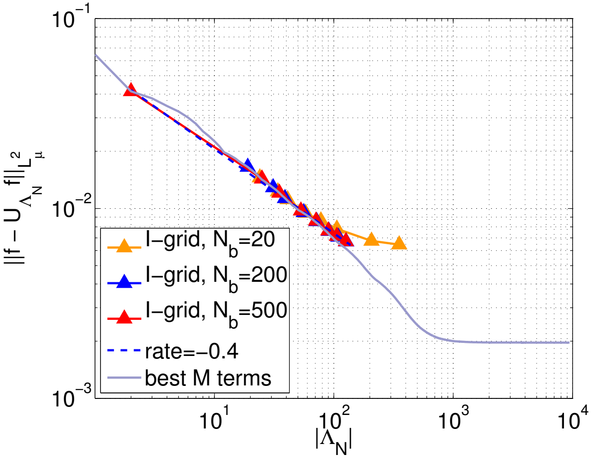
- Labels show the number of activated random variables
- Similar rate to before $\Rightarrow$ growth of points linear in $|\Lambda_N|$
- best-N-terms obtained by converting sparse grid into Hermite polynomials with **`convert_to_modal`** and sorting the coefficients

The importance of the buffer size N_b



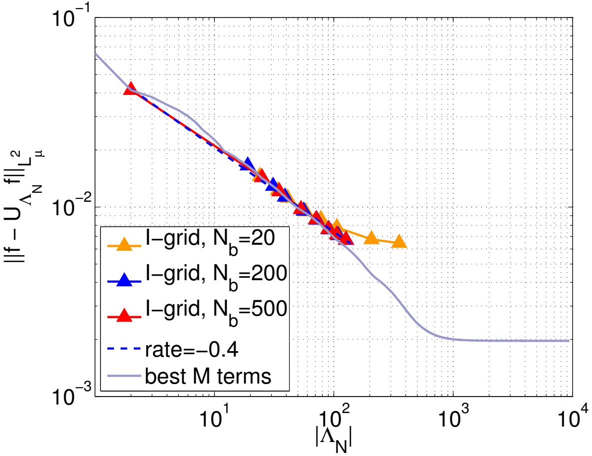
- Levy-Ciesielki expansion of $\log a(x, \boldsymbol{\xi})$ uses Faber-Schauder basis (primitive of the Haar functions): $|\phi_m|$ are not monotone decreasing (contrary to KL)

The importance of the buffer size N_b



- Levy-Ciesielki expansion of $\log a(x, \boldsymbol{\xi})$ uses Faber-Schauder basis (primitive of the Haar functions): $|\phi_m|$ are not monotone decreasing (contrary to KL)
- use large N_b, or convengence will stagnate

The importance of the buffer size N_b



- Levy-Ciesielki expansion of $\log a(x, \boldsymbol{\xi})$ uses Faber-Schauder basis (primitive of the Haar functions): $|\phi_m|$ are not monotone decreasing (contrary to KL)
- use large N_b, or convengence will stagnate
- a-posteriori grid departs from best-M-terms: unsignificant modes have been added to the a-posteriori grid.

Ex. 2) Geochemical compaction problem

How do we get from from sand on the bottom of the sea to sedimentary basins?

Ex. 2) Geochemical compaction problem

How do we get from from sand on the bottom of the sea to sedimentary basins?

1. Mechanical forces (water column and new sediments) push sand grains closer and downward towards the center the earth, water is squeezed out. The deeper layers thus get warmer.

Ex. 2) Geochemical compaction problem

How do we get from from sand on the bottom of the sea to sedimentary basins?

1. Mechanical forces (water column and new sediments) push sand grains closer and downward towards the center the earth, water is squeezed out. The deeper layers thus get warmer.
2. As temperature increases, chemical reactions which glue grains together are triggered (quartz precipitation).

Ex. 2) Geochemical compaction problem

How do we get from from sand on the bottom of the sea to sedimentary basins?

1. Mechanical forces (water column and new sediments) push sand grains closer and downward towards the center the earth, water is squeezed out. The deeper layers thus get warmer.
2. As temperature increases, chemical reactions which glue grains together are triggered (quartz precipitation).
3. This process is described by a set of coupled, time-dependent, non-linear, monodimensional (depth-only) PDEs.

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)
  - $\phi_k = \Phi_k(\mathbf{y}) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for $k = 1, \ldots, K$,

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)
  - $\phi_k = \Phi_k(\mathbf{y}) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for $k = 1, \dots, K$,
  - $T_\ell = \mathrm{T}_\ell(\mathbf{y}) + \eta_\ell, \qquad \eta_\ell \sim \mathcal{N}(0, s^2)$ i.i.d. for $\ell = 1, \dots, L$

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)
  - $\phi_k = \Phi_k(\mathbf{y}) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for $k = 1, \ldots, K$,
  - $T_\ell = \mathrm{T}_\ell(\mathbf{y}) + \eta_\ell, \qquad \eta_\ell \sim \mathcal{N}(0, s^2)$ i.i.d. for $\ell = 1, \ldots, L$
- $\Phi_k(\mathbf{y})$, $\mathrm{T}_\ell(\mathbf{y})$ are model values

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)
  - $\phi_k = \Phi_k(\mathbf{y}) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for $k = 1, \ldots, K$,
  - $T_\ell = \mathrm{T}_\ell(\mathbf{y}) + \eta_\ell, \qquad \eta_\ell \sim \mathcal{N}(0, s^2)$ i.i.d. for $\ell = 1, \ldots, L$
- $\Phi_k(\mathbf{y})$, $\mathrm{T}_\ell(\mathbf{y})$ are model values
- σ, s unknown! account for both measure and model error

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)
  - $\phi_k = \Phi_k(\mathbf{y}) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for $k = 1, \ldots, K$,
  - $T_\ell = \mathrm{T}_\ell(\mathbf{y}) + \eta_\ell, \qquad \eta_\ell \sim \mathcal{N}(0, s^2)$ i.i.d. for $\ell = 1, \ldots, L$
- $\Phi_k(\mathbf{y})$, $\mathrm{T}_\ell(\mathbf{y})$ are model values
- σ, s unknown! account for both measure and model error
- Square errors: $J_\Phi(\mathbf{y}) = \sum_{k=1}^{K}(\phi_k - \Phi_k(\mathbf{y}))^2, \quad J_{\mathrm{T}}(\mathbf{y}) = \sum_{\ell=1}^{L}(v_\ell - \mathrm{T}_\ell(\mathbf{y}))^2$

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)
  - $\phi_k = \Phi_k(\mathbf{y}) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for $k = 1, \ldots, K$,
  - $T_\ell = \mathrm{T}_\ell(\mathbf{y}) + \eta_\ell, \qquad \eta_\ell \sim \mathcal{N}(0, s^2)$ i.i.d. for $\ell = 1, \ldots, L$
- $\Phi_k(\mathbf{y})$, $\mathrm{T}_\ell(\mathbf{y})$ are model values
- σ, s unknown! account for both measure and model error
- Square errors: $J_\Phi(\mathbf{y}) = \sum_{k=1}^{K} (\phi_k - \Phi_k(\mathbf{y}))^2, \quad J_{\mathrm{T}}(\mathbf{y}) = \sum_{\ell=1}^{L} (v_\ell - \mathrm{T}_\ell(\mathbf{y}))^2$
- $NLL(\mathbf{y}, \sigma, s) = \dfrac{J_\Phi(\mathbf{y})}{\sigma^2} + \dfrac{J_{\mathrm{T}}(\mathbf{y})}{s^2} + K \log \sigma^2 + L \log s^2 + (K + L) \log \sqrt{2\pi}$

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)
  - $\phi_k = \Phi_k(\mathbf{y}) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for $k = 1, \ldots, K$,
  - $T_\ell = \mathrm{T}_\ell(\mathbf{y}) + \eta_\ell, \qquad \eta_\ell \sim \mathcal{N}(0, s^2)$ i.i.d. for $\ell = 1, \ldots, L$
- $\Phi_k(\mathbf{y})$, $\mathrm{T}_\ell(\mathbf{y})$ are model values
- σ, s unknown! account for both measure and model error
- Square errors: $J_\Phi(\mathbf{y}) = \sum_{k=1}^{K} (\phi_k - \Phi_k(\mathbf{y}))^2, \quad J_{\mathrm{T}}(\mathbf{y}) = \sum_{\ell=1}^{L} (v_\ell - \mathrm{T}_\ell(\mathbf{y}))^2$
- $NLL(\mathbf{y}, \sigma, s) = \dfrac{J_\Phi(\mathbf{y})}{\sigma^2} + \dfrac{J_{\mathrm{T}}(\mathbf{y})}{s^2} + K \log \sigma^2 + L \log s^2 + (K + L) \log \sqrt{2\pi}$
- For later use $\lambda = \dfrac{\sigma^2}{s^2}$

Maximum Likelihood Inversion

- Let $\mathbf{y} \in \Gamma$ be the vector of uncertain parameters of the compaction PDE system.
- Noisy porosity and temperature data (in depth)
  - $\phi_k = \Phi_k(\mathbf{y}) + \varepsilon_k, \qquad \varepsilon_k \sim \mathcal{N}(0, \sigma^2)$ i.i.d. for $k = 1, \ldots, K$,
  - $T_\ell = \mathrm{T}_\ell(\mathbf{y}) + \eta_\ell, \qquad \eta_\ell \sim \mathcal{N}(0, s^2)$ i.i.d. for $\ell = 1, \ldots, L$
- $\Phi_k(\mathbf{y})$, $\mathrm{T}_\ell(\mathbf{y})$ are model values
- σ, s unknown! account for both measure and model error
- Square errors: $J_\Phi(\mathbf{y}) = \sum_{k=1}^{K} (\phi_k - \Phi_k(\mathbf{y}))^2, \quad J_\mathrm{T}(\mathbf{y}) = \sum_{\ell=1}^{L} (v_\ell - \mathrm{T}_\ell(\mathbf{y}))^2$
- $NLL(\mathbf{y}, \sigma, s) = \frac{J_\Phi(\mathbf{y})}{\sigma^2} + \frac{J_\mathrm{T}(\mathbf{y})}{s^2} + K \log \sigma^2 + L \log s^2 + (K + L) \log \sqrt{2\pi}$
- For later use $\lambda = \frac{\sigma^2}{s^2}$
- Underlying fundamental question: is it better to have porosity or temperature data?

Maximum Likelihood Inversion – details

Then, a MLE procedure to estimate $\mathbf{y}, \sigma, s$ is

Eval: `[phi_values,T_values] = evaluate_on_sparse_grids(@model,S,Sr,..)`

Maximum Likelihood Inversion – details

Then, a MLE procedure to estimate $\mathbf{y}, \sigma, s$ is

Eval:
```
[phi_values,T_values] = evaluate_on_sparse_grids(@model,S,Sr,..)
```

For $\lambda = \lambda_1, \ldots, \lambda_N$

Maximum Likelihood Inversion – details

Then, a MLE procedure to estimate $\mathbf{y}, \sigma, s$ is

Eval:
```
[phi_values,T_values] = evaluate_on_sparse_grids(@model,S,Sr,..)
```

For $\lambda = \lambda_1, \ldots, \lambda_N$

1. $\mathbf{y}^* = \mathrm{argmin}_{\mathbf{y}\in\Gamma} J_\Phi + \lambda J_\mathrm{T}$ (i.e., $\mathrm{argmin}_{\mathbf{y}\in\Gamma} NLL$ for fixed σ, s)

```
% define @-fun to call interpolate_on_sparse_grid(S,Sr,phi_values,p)
f = @(y) eval_J_phi(S,Sr,phi_values,phi_data,y) + eval_J_T ..
y_MLE = fminsearch(f,p0);
```

Maximum Likelihood Inversion – details

Then, a MLE procedure to estimate $\mathbf{y}, \sigma, s$ is

Eval:

```
[phi_values,T_values] = evaluate_on_sparse_grids(@model,S,Sr,..)
```

For $\lambda = \lambda_1, \ldots, \lambda_N$

1. $\mathbf{y}^* = \mathrm{argmin}_{\mathbf{y} \in \Gamma} J_\Phi + \lambda J_\mathrm{T}$ (i.e., $\mathrm{argmin}_{\mathbf{y} \in \Gamma} NLL$ for fixed σ, s)

```
% define @-fun to call interpolate_on_sparse_grid(S,Sr,phi_values,p)
f = @(y) eval_J_phi(S,Sr,phi_values,phi_data,y) + eval_J_T ..
y_MLE = fminsearch(f,p0);
```

2. $\sigma = \dfrac{J_\Phi(\mathbf{y}^*) + \lambda J_\mathrm{T}(\mathbf{y}^*)}{K+L}, s^2 = \dfrac{\sigma^2}{\lambda}$

Maximum Likelihood Inversion – details

Then, a MLE procedure to estimate $\mathbf{y}, \sigma, s$ is

Eval:
```
[phi_values,T_values] = evaluate_on_sparse_grids(@model,S,Sr,..)
```

For $\lambda = \lambda_1, \ldots, \lambda_N$

1. $\mathbf{y}^* = \mathrm{argmin}_{\mathbf{y}\in\Gamma} J_\Phi + \lambda J_\mathrm{T}$ (i.e., $\mathrm{argmin}_{\mathbf{y}\in\Gamma} NLL$ for fixed σ, s)

```
% define @-fun to call interpolate_on_sparse_grid(S,Sr,phi_values,p)
f = @(y) eval_J_phi(S,Sr,phi_values,phi_data,y) + eval_J_T ..
y_MLE = fminsearch(f,p0);
```

2. $\sigma = \dfrac{J_\Phi(\mathbf{y}^*) + \lambda J_\mathrm{T}(\mathbf{y}^*)}{K+L}, s^2 = \dfrac{\sigma^2}{\lambda}$
3. $KIC = NLL - N \log 2\pi - \ln det(C)$
   - $C = \sigma^2 (D_\mathbf{y}\phi^\top D_\mathbf{y}\phi + \lambda D_\mathbf{y} T^\top D_\mathbf{y} T)^{-1}$
   - $D_\mathbf{y}\phi, D_\mathbf{y} T$ Jacobian matrices of ϕ and T wrt to $\mathbf{y}$, evaluated at $\mathbf{y}^*$ (sensitivity)

```
for i=1:nb_z_nodes
  Jac_phi(i,:) = derive_sparse_grid(S,Sr,phi_values(i,:),y_MLE);
end
```

Maximum Likelihood Inversion – details

Then, a MLE procedure to estimate $\mathbf{y}, \sigma, s$ is

Eval:
```
[phi_values,T_values] = evaluate_on_sparse_grids(@model,S,Sr,..)
```

For $\lambda = \lambda_1, \ldots, \lambda_N$

1. $\mathbf{y}^* = \text{argmin}_{\mathbf{y}\in\Gamma} J_\Phi + \lambda J_\mathrm{T}$ (i.e., $\text{argmin}_{\mathbf{y}\in\Gamma} NLL$ for fixed σ, s)

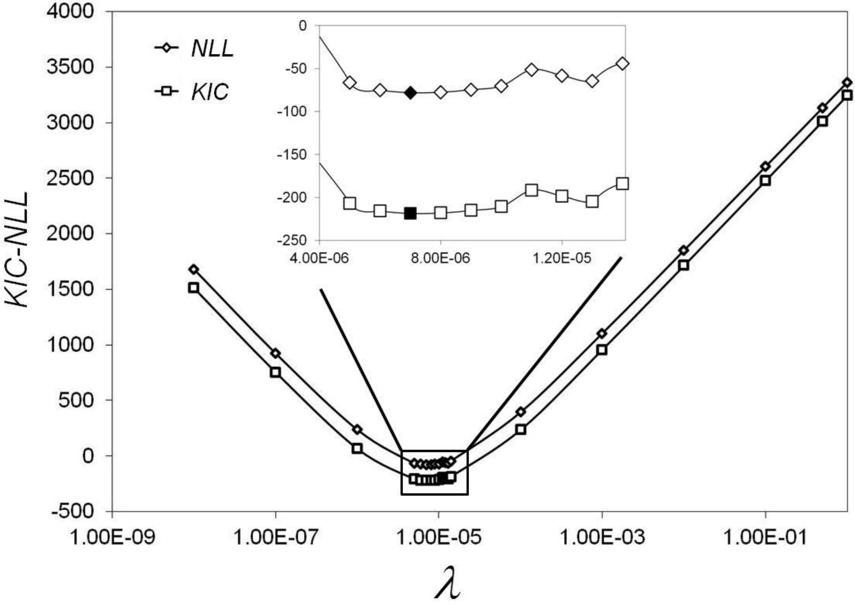
```
% define @-fun to call interpolate_on_sparse_grid(S,Sr,phi_values,p)
f = @(y) eval_J_phi(S,Sr,phi_values,phi_data,y) + eval_J_T ..
y_MLE = fminsearch(f,p0);
```

2. $\sigma = \dfrac{J_\Phi(\mathbf{y}^*) + \lambda J_\mathrm{T}(\mathbf{y}^*)}{K+L}, s^2 = \dfrac{\sigma^2}{\lambda}$

3. $KIC = NLL - N \log 2\pi - \ln det(C)$
   - $C = \sigma^2 (D_\mathbf{y}\phi^\top D_\mathbf{y}\phi + \lambda D_\mathbf{y} T^\top D_\mathbf{y} T)^{-1}$
   - $D_\mathbf{y}\phi, D_\mathbf{y} T$ Jacobian matrices of ϕ and T wrt to $\mathbf{y}$, evaluated at $\mathbf{y}^*$ (sensitivity)

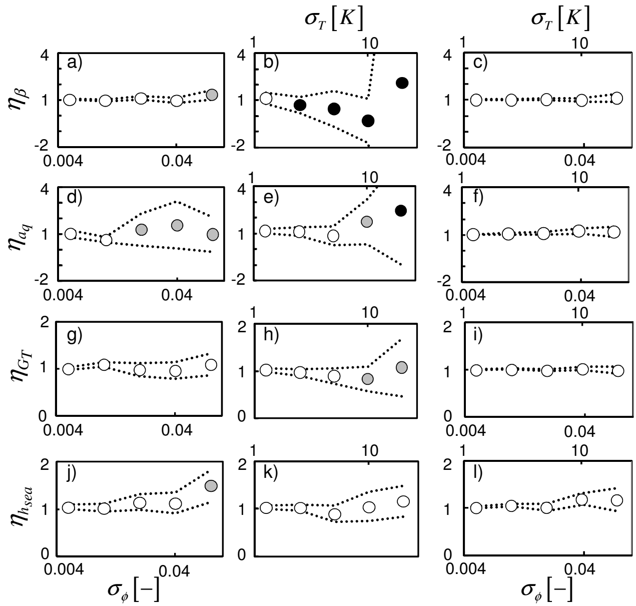
```
for i=1:nb_z_nodes
  Jac_phi(i,:) = derive_sparse_grid(S,Sr,phi_values(i,:),y_MLE);
end
```

Choose $(\sigma, s, \mathbf{y})$ that yield the minimum KIC



NLL
KIC
KIC-NLL
λ
4000
3500
3000
2500
2000
1500
1000
500
0
-500
1.00E-09
1.00E-07
1.00E-05
1.00E-03
1.00E-01
0
-50
-100
-150
-200
-250
4.00E-06
8.00E-06
1.20E-05

Results



$\sigma_T\,[K]$
$\sigma_T\,[K]$
a)
b)
c)
d)
e)
f)
g)
h)
i)
j)
k)
l)
η_β
η_{aq}
η_{GT}
η_{hsea}
4
1
-2
2
0
0.004
0.04
10
$\sigma_\phi\,[-]$
$\sigma_\phi\,[-]$

Outline

1. Basic data structure
2. Main features
3. Numerical examples
4. Conclusions

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;
- geared towards UQ, can handle a variety of knots families and distribution.

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- geared towards UQ, can handle a variety of knots families and distribution.

Bibliography

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- geared towards UQ, can handle a variety of knots families and distribution.

Bibliography

- O. G. Ernst, B. Sprungk, L. Tamellini. *Convergence of Sparse Collocation for Functions of Countably Many Gaussian Random Variables (with Application to Lognormal Elliptic Diffusion Problems).* SIAM Journal on Numerical Analysis, 2018.

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- geared towards UQ, can handle a variety of knots families and distribution.

Bibliography

- O. G. Ernst, B. Sprungk, L. Tamellini. *Convergence of Sparse Collocation for Functions of Countably Many Gaussian Random Variables (with Application to Lognormal Elliptic Diffusion Problems).* SIAM Journal on Numerical Analysis, 2018.
- G. Porta, L. Tamellini, V. Lever, M. Riva. *Inverse modeling of geochemical and mechanical compaction in sedimentary basins through polynomial chaos expansion.* Water Resources Research, 2014.

Concluding remarks

- high-level Matlab software, based on combination technique form of sparse grids;
- geared towards UQ, can handle a variety of knots families and distribution.

Bibliography

- O. G. Ernst, B. Sprungk, L. Tamellini. *Convergence of Sparse Collocation for Functions of Countably Many Gaussian Random Variables (with Application to Lognormal Elliptic Diffusion Problems).* SIAM Journal on Numerical Analysis, 2018.
- G. Porta, L. Tamellini, V. Lever, M. Riva. *Inverse modeling of geochemical and mechanical compaction in sedimentary basins through polynomial chaos expansion.* Water Resources Research, 2014.
- J. Martnez-Frutos, F. Periago Esparza, *Optimal Control of PDEs under Uncertainty,* Springer 2018. Uses Matlab Sparse Grids Kit