1 Introduction

This document presents the MATLAB ZS+G module. ZS+G is an application for creating multidimensional grids (full tensor grids and sparse grids) in order to create optimal experimental designs for surrogate models. The user is introduced to ZS+G in this document, which is divided into the following sections:

I. System requirements for ZS+G.

II. Installation and uninstallation procedure.

III. CLI interface usage

IV. Minimal examples

V. ZS+G classes repository

2 Software requirements

ZS+G requires MATLAB to run. Please go to MathWorks to download MATLAB.

ZS+G is a standalone application for generating unit grids on the interval $[-1, 1]^d$ but requires UQLab for mapping the grid on natural supports such as random vectors. Some features of ZS+G will not work without the installation of UQLab. That is why, its installation is strongly recommended before using ZS+G. The user can download UQLab for free here. Please follow then the installation procedure of UQLab.

3 Installation of ZS+G

The following installation procedure applies :

I. Go to ZS+G.git to download the .zip file containing ZS+G.

II. Unzip the file in Program Files folder on your machine (recommended) or in another chosen folder.

III. Open MATLAB in administrator mode and select `yourfolder\ZS_G\core` as the Current Folder in MATLAB.

IV. Type `ZS_G_install` in the MATLAB console.

4 Uninstallation of ZS+G

The following uninstallation procedure applies :

I. Open MATLAB in administrator mode and select `yourfolder\ZS_G\core` as the Current Folder in MATLAB.

II. Type `ZS_G_uninstall` in the MATLAB console.

5 MATLAB class repository

5.1 ZS_Points | Class

	Class properties				
Name	Sj	nestedFlag	Family	M	N
Class	`cell`	`logical`	`char`	`double`	`double`
Size	$(1\times:)$	(1×1)		(1×1)	(1×1)

5.1.1 `ZS_Points` (family , selection , expr , bounds) | constructor

Error handling : YES

This class creates a `ZS_Points` object. Sj ($=S^j$) is a set of subsets containing the unidimensional 1d-vector. nestedFlag is a logical value which indicates if the subsets contained in S^k are nested. Family is the name of the nodes family given by the user. M is the number of subsets in S^j. N is the total number of elements in S^j.

	Input args				Output args
Name	family	selection	expr	bounds	self
	•	•	⊡	⊡	
Class	`char` 'linspace' 'chebyshev_1' 'chebyshev_2' 'legendre' 'lobatto'	$(1\times:)$ `double`	`char`	`logical`	`ZS_Points`
Default value			'2^n+1'	true	

5.1.2 `.print_set` ( ) | class method

Error handling : NO

This function makes a scatter plot of the subsets of S^j. X-axis represents the coordinates over the closed interval [-1 1] and Y-axis denotes the index i of S_i^j

Example :

`obj = ZS_Points('chebishev_2', 5)`	→ `ZS_Points`
`obj.print_set`	→ figure with a scatter plot

5.1.3 .`get_family` ( ) | static method

Error handling : NO

This function returns the current implemented point families of the ZS_Points class.

5.1.4 .`convert_pts` (selection , expr) | static method

Error handling : YES

This function is powerful. It maps the mathematical expression(s) in **expr** element-wise or column-wise over **selection**. It have been adapted to flexibly behave against the user inputs. It is particularly suitable for generating growing sequences of integers used then to generate a set S^j with subsets of unidimensional vectors. Be careful : when **selection** is not a vector of size $(: \times 1)$ or $(1\times :)$, i.e. **selection** is a matrix, then the mapping is performed column-wise. In this case, **expr** must be a cell array with size $(1 \times k)$ where k is the number of columns of **selection**.

	Input args		Output args
Name	selection	expr	new_selection
	•	⊡	
Class	`double`	`char` or `cell`	`double`
Size	$(: \times :)$	or $(1\times :)$	$(: \times :)$
Default value		'n'	

Example :

ZS_Points.convert_pts$(1:5)$	$\rightarrow [1\ 2\ 3\ 4\ 5]$
ZS_Points.convert_pts$(1:5,$ '2^n'$)$	$\rightarrow [2\ 4\ 8\ 16\ 32]$
ZS_Points.convert_pts$(5:1,$ '2^n'$)$	$\rightarrow [2\ 4\ 8\ 16\ 32]^\intercal$
`pts = randi(5,5,2)`	$\rightarrow \begin{bmatrix} 1 & 4 & 2 & 4 & 5 \\ 3 & 4 & 4 & 3 & 4 \end{bmatrix}^\intercal$
ZS_Points.convert_pts(pts, '2^n')	$\rightarrow \begin{bmatrix} 2 & 16 & 4 & 16 & 32 \\ 8 & 16 & 16 & 8 & 16 \end{bmatrix}^\intercal$
ZS_Points.convert_pts(pts, {'n', '2^n'})	$\rightarrow \begin{bmatrix} 1 & 4 & 2 & 4 & 5 \\ 8 & 16 & 16 & 8 & 16 \end{bmatrix}^\intercal$

5.1.5 .remove_bounds (vec_1d) | static method

Error handling : YES

This function removes the points -1 and 1 of a unidimensional vector **vec_1d**.

	Input args	Output args
Name	vec_1d	new_selection
	•	
Class	`double`	`double`
Size	$(: \times :)$	$(: \times :)$

Example :

`ZS_Points.remove_bounds`$([-1\ 0\ 1]) \rightarrow [0]$

5.1.6 .check_nested (S) | static method

Error handling : YES

This function tests if the set S^k contains nested subsets. If **S** is not a cell, the *nestedness* has no sense.

	Input args	Output args
Name	S	nestedFlag
	•	
Class	`double` or `cell`	`logical`
Size	$(: \times :)$ or $(1 \times :)$	(1×1)

Example :

`ZS_Points.check_nested`('a') $\rightarrow$ false

`ZS_Points.check_nested`$([-1\ 0\ 1\ 1])$ $\rightarrow$ false

`ZS_Points.check_nested`$(\{[0], [-1\ 0\ 1], [-1\ \ -0.5\ 0\ 0.5\ 1]\})$ $\rightarrow$ true

`ZS_Points.check_nested`({"a", ["a" "b"], ["b" "c"]}) $\rightarrow$ false

5.1.7 .get_pts_linspace (selection , bounds) | static method

This function creates a set of unidimensional vectors. The set is represented in MATLAB as a cell array containing M unidimensional vectors whose sizes are driven within generator vector **selection**. If **selection** is a vector of size = 1, the function returns only a unidimensional vector. Second output argument is a logical value if the set is nested or not. This flag is always false if **selection** is a vector of size = 1. Option argument **bounds** controls the boundary points of the domain [-1 1] (true = keep the boundary points, false = delete the boundary points).

	Input args		Output args	
Name	selection	bounds	nodes	nestedFlag
	•	⊡		
Class	double	logical	cell	logical
Size	$(1 \times \mathrm{M})$	(1×1)	$(1 \times \mathrm{M})$	(1×1)
Default value		true		

Example :

`ZS_Points.get_pts_linspace(1)`	$\rightarrow [0]$	false
`ZS_Points.get_pts_linspace(3)`	$\rightarrow [-1\ 0\ 1]$	false
`ZS_Points.get_pts_linspace(1:3)`	$\rightarrow \{[0]\ ,\ [-\frac{2}{3}\ \frac{2}{3}]\ ,\ [-1\ 0\ 1]\}$	false
`ZS_Points.get_pts_linspace(1:3, true)`	$\rightarrow \{[0]\ ,\ [-\frac{2}{3}\ \frac{2}{3}]\ ,\ [0]\}$	false

5.1.8 .get_pts_chebyshev_1 (selection , ∼) | static method

Explanations in Point 5.1.7 hold. This function returns the Chebyshev nodes of the first kind. Since they do not have any boundary points, second argument is unused.

Example :

`ZS_Points.get_pts_chebyshev_1(3)`	$\rightarrow [-\frac{\sqrt{3}}{2}\ 0\ \frac{\sqrt{3}}{2}]$	false
`ZS_Points.get_pts_chebyshev_1(1:3)`	$\rightarrow \{[0]\ ,\ [-\frac{1}{\sqrt{2}}\ \frac{1}{\sqrt{2}}]\ ,\ [-\frac{\sqrt{3}}{2}\ 0\ \frac{\sqrt{3}}{2}]\}$	false

5.1.9 .get_pts_chebyshev_2 (selection , bounds) | static method

Explanations in Point 5.1.7 hold. This function returns the Chebyshev nodes of the second kind.

Example :

`ZS_Points.get_pts_chebyshev_2`(3)	$\rightarrow [-1\ 0\ 1]$	false
`ZS_Points.get_pts_chebyshev_2`$([1\ 3\ 5])$	$\rightarrow \{[0]\ ,\ [-1\ 0\ 1]\ ,\ [-1\ -\frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}\ 1]\}$	true
`ZS_Points.get_pts_chebyshev_2`$([1\ 3\ 5], \text{false})$	$\rightarrow \{[0]\ ,\ [0]\ ,\ [-\frac{1}{\sqrt{2}}\ 0\ \frac{1}{\sqrt{2}}]\}$	true

5.1.10 .get_pts_legendre (selection , ∼) | static method

Explanations in Point 5.1.7 hold. This function returns the Gauss-Legendre nodes. Since they do not have any boundary points, second argument is unused.

5.1.11 .get_pts_lobatto (selection , bounds) | static method

Explanations in Point 5.1.7 hold. This function returns the Gauss-Legendre-Lobatto nodes.

5.2 ZS_SparseGrid | Class

	Class properties						
Name	Unit_grid	Class	Level	Dimensions	nestedFLag	Basis	Mapping
Class	double	char	double	double	logical	struct	struct
Size	$(: \times d)$		(1×1)	(1×2)	(1×1)		

5.2.1 ZS_SparseGrid (Validation) | constructor

This class creates a ZS_SparseGrid object. test affafefewgegweg