Universität Stuttgart
Institute of Applied Analysis and Numerical Simulation

SPINTERP V2.1
Piecewise multilinear hierachical sparse grid interpolation in Matlab
DOCUMENTATION

Andreas Klimke

Version from April 6, 2005

Contents

1 Introduction

To recover or approximate smooth multivariate functions, sparse grids are superior to full grids due to a significant reduction of the required support nodes. The order of the convergence rate in the maximum norm is preserved up to a logarithmic factor. Three piecewise multilinear hierarchical interpolation schemes are included in the software package `spinterp` for MATLAB. In the following, we describe the basic usage and advanced features. Theoretical and algorithmic aspects are discussed in the homonymous article [3].

To make the tool as easy as possible to use, we decided to follow the approach of [5]. All sparse grid interpolation routines can be called in the same way, and providing additional options is optional.

This documentation is organized as follows. Section 2 contains a brief function reference. In section 3, we provide two additional examples illustrating advanced features.

2 Function reference

The following functions comprise the implementation. For further details on the syntax of each function, please use `help <function_name>` within MATLAB.

init.m

Calling this file will add the relevant directories containing the sparse grid algorithms to your MATLAB path. Furthermore, it displays a list of available demonstration m-files.

spvals.m

Determines the hierarchical surpluses of the sparse grid interpolant for a `d`-variate function `fun` over a specified interval box. By default, the Clenshaw-Curtis grid is used. The function `fun` may be an inline function, a function handle, or a function file, and must accept `d` parameters (unless other parameters `p1`, `p2`, ... (see below) are passed to the function). The following syntax options are available:

`z = spvals(fun, d)`: Computes the hierarchical surpluses of a function `fun`. The grid is computed over the `d`-dimensional unit cube $[0,1]^d$.

`z = spvals(fun, d, range)`: If the objective interpolation box is not the unit cube, you may specify the interval `range` of each input parameter as a `d`×`2` matrix. E.g. if the range of parameter 1 is $[0,2]$, and the range of parameter 2 is $[1,4]$, you would set `range` to `[0 2; 1 4]`.

`z = spvals(fun, d, range, options, p1, p2, ...)`: In addition to the simple syntax, you may pass an `options` structure to `spvals`, which can be used to change the default sparse grid interpolation parameters. Create the options argument with the `spset` function. For an explanation of the available options, please refer to Table 2.1. Any input parameters after the options structure are passed as additional parameters to the objective function `fun`.

As result, `spvals` returns a structure containing the cell array of hierarchical surpluses as well as some statistical information (see Table 2.2 for more information). You may access each property with `z.<propertyName>`.

spinterp.m

Once the sparse representation of a function has been determined with `spvals`, one can compute interpolated values with `spinterp` for any point $\mathtt{y} = (y_1, \ldots, y_d)$ within the specified range of the input parameters. The syntax is

```
ip = spinterp(z, y1, ..., yd)
```

Here is a simple example for computing the sparse grid data for a function f with $d = 3$. Then, the interpolated value at the point $(0.5, 0.2, 0.2)$ and the absolute interpolation error is determined.

Example 2.1.

```
f = inline('x.^2 + y.^2 - 2.*z');
z = spvals(f,3);
f_interp = spinterp(z, 0.5, 0.2, 0.2)
  f_interp =
```

Table 2.1: Available options configurable with `spset`.

Property	*Value*	*Description*
`GridType`	String	The grid type. The default type is `'Clenshaw-Curtis'`, other possibly values are `'Maximum'` and `'NoBoundary'`.
`RelTol`	Scalar	A relative error tolerance that applies to all hierarchical surpluses w^k of the current deepest level k of the sparse grid interpolation formula. The default value is 10^{-2} (1 % accuracy). The interpolation depth level k is increased until the absolute values of all w^k are less than $$\max\left(\mathrm{RelTol}\cdot(\max(\mathrm{fvals})-\min(\mathrm{fvals}))\,,\mathrm{AbsTol}\right),$$ with `fvals` containing all results evaluating `fun` up to that point.
`AbsTol`	Scalar	Absolute error tolerance. The default value is 10^{-6}.
`Vectorized`	on | off	Indicates if `fun` is available for vectorized evaluation. The default value is `'off'`. Vectorized coding of `fun` can signifi cantly reduce the computation time used by `spvals`.
`MinDepth`	Integer	Minimum number of levels $n = q - d$ to compute (default is 2).
`MaxDepth`	Integer	Maximum number of levels to compute (default is 8).
`VariablePositions`	Vector	Sometimes it is useful to change the order of inputs to `fun`. Please refer to `help spset` for additional information.
`NumberOfOutputs`	Integer	If `fun` produces multiple outputs (all must be scalar), indicate this here to perform the sparse grid computation for many output variables at once. Also see the example `spdemovarout.m`.
`PrevResults`	Structure	An existing result structure obtained from `spvals` may be provided to further refi ne an existing sparse grid.

Table 2.2: `spvals` output properties.

Property	*Value*	*Description*
`vals`	Cell array	Contains `maxLevel`+1 matrices of the hierarchical surpluses of each interpolation depth level.
`gridType`	String	The grid type.
`d`	Integer	The problem dimension.
`range`	Matrix	The range of the input parameters. An empty matrix indicates the interval box $[0,1]^d$.
`maxLevel`	Integer	The actual interpolation depth of the sparse grid representation. It depends on the requested minimum/maximum interpolation depth `MinDepth`/`MaxDepth` and the requested error tolerances `RelTol`/`AbsTol`.
`estRelError`	Scalar	The estimated relative error e with respect to `fevalRange`: $$e = \frac{\max(\lvert w^k\rvert)}{\max(\mathrm{fevalRange})-\min(\mathrm{fevalRange})}$$
`fevalRange`	Matrix	The minimum and maximum of all function values encountered during the evaluation of `fun` at the sparse grid support points. Each row of the matrix contains the results of one output parameter.
`nPoints`	Integer	The number of grid points.
`fevalTime`	Scalar	The amount of time spent evaluating `fun` at the sparse grid points.
`surplusCompTime`	Scalar	Time spent computing the hierarchical surpluses.

```
     -0.1063
error = abs(f(0.5, 0.2, 0.2) - f_interp)
  error =
      0.0037
```

You may also call `spinterp` for multiple points at once, with `y1`, ..., `yn` containing vectors or matrices of equal dimension, i.e. `size(y1)` $= \ldots =$ `size(yd)`.

spset.m

The `spset` function creates an options structure that you can supply to the `spvals` function (see Table 2.1). `spset` accepts property name/property value pairs using the syntax

```
options = spset('name1', value1, 'name2', value2, ...)
```

The function is case-insensitive. Called with no input parameters, `spset` displays the possible values and the defaults. You may also modify an existing options structure.

Example 2.2. To compute a sparse grid interpolant of a two-dimensional function $f(x,y) = \exp(x^2+y^2)$ in $[0,1]^2$ using the maximum-norm-based grid H^{M}, vectorized processing, and a stringent relative error tolerance of 10^{-4}, proceed as follows:

```
f = inline('exp(x.^2␣+␣y.^2)');
options = spset('GridType', 'Maximum', 'Vectorized', 'on', ...
                'RelTol', 1e-4);
z = spvals(f, 2, [], options);
```

spget.m

You may query an existing options structure with `spget`. The syntax is

```
value = spget(options, 'name')
```

If the property `name` exists, the current value is returned, otherwise, an empty matrix `[]`.

spdim.m

Computes the number of points of the sparse grid $H_{q,d}$ (by default the Clenshaw-Curtis grid H^{CC}). You may use an options structure to select other grid types. The syntax options are

```
spdim(n,d)
spdim(n,d,options)
```

where `n` denotes the interpolation depth $n = q - d$ of the corresponding interpolation formula $A_{q,d}(f)$. For example, to compute the number of points of $H^{\mathrm{CC}}_{17,10}$, use the syntax

```
spdim(7,10)
  ans =
      652065
```

with $n = q - d = 7$.

`spgrid.m`

Explicitly computes the coordinates of the points of a single level of a sparse grid (the Clenshaw-Curtis grid by default). This function is internally used by `spvals` to compute the grid points that f needs to be evaluated for, and is usually not required to be called by the user.

Example 2.3. Let us explicitly compute and plot the grid points of $\Delta H^{\text{NB}}_{n+d,d}$ for $d = 2, n = 2$.

```
x = spgrid(2,2,spset('GridType','NoBoundary'))
  x =
     0.1250    0.5000
     0.3750    0.5000
     0.6250    0.5000
     0.8750    0.5000
     0.2500    0.2500
     0.7500    0.2500
     0.2500    0.7500
     0.7500    0.7500
     0.5000    0.1250
     0.5000    0.3750
     0.5000    0.6250
     0.5000    0.8750

plot(x(:,1),x(:,2),'k.'); axis equal; axis tight;
```

`plotgrid.m`

The `plotgrid` command serves as to quickly visualize two- and three- dimensional grids of specified depth. The input arguments are, as with `spgrid.m` and `spdim.m`, the interpolation depth $n = q - d$, the dimension d, and an optional options structure created with `spset`.

Example 2.4. The following command plots the Clenshaw-Curtis grid $H^{\text{CC}}_{6+2,2}$, as shown in Fig. 2.1.

```
plotgrid(6,2);
```

Note: The included demonstration file `cmpgrids.m` compares the three available multilinear grid types in two and three dimensions.

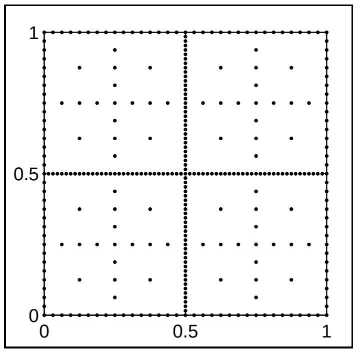

Figure 2.1: Clenshaw-Curtis grid $H^{\text{CC}}_{8,2}$

3 Examples

In this section, two examples are provided. Further demonstrations are included with the software (see `init.m`).

A simple two-dimensional function

The following script represents a simple example in two dimensions, including multiple evaluations of the interpolating function $A_{q,d}(f)$ through a vectorized call of `spinterp`. This example is also included in the software package as `spdemo.m`. The graphical output is shown in Fig. 3.1.

```
% Some objective function f
f = inline('1./((x*2-0.3).^4 +(y*3-0.7).^2+1)');

% Define problem dimension
d = 2;

% Create full grid for plotting
gs = 33;
[X,Y] = meshgrid(linspace(0,2,gs),linspace(-1,1,gs));

% Set options: Switch vectorized processing on.
options = spset('Vectorized', 'on');

% Compute sparse grid weights over domain [0,2]x[-1,1]
z = spvals(f, d, [0 2; -1 1], options);

% Compute interpolated values at full grid
ip = spinterp(z, X, Y);

% Plot original function, interpolation, and error
subplot(1,3,1);
mesh(X,Y,f(X,Y));
title('original');
subplot(1,3,2);
mesh(X,Y,ip);
title('interpolated');
subplot(1,3,3);
mesh(X,Y,abs(f(X,Y)-ip));
title('absolute error');

disp('Sparse grid representation of the function:');
z
```

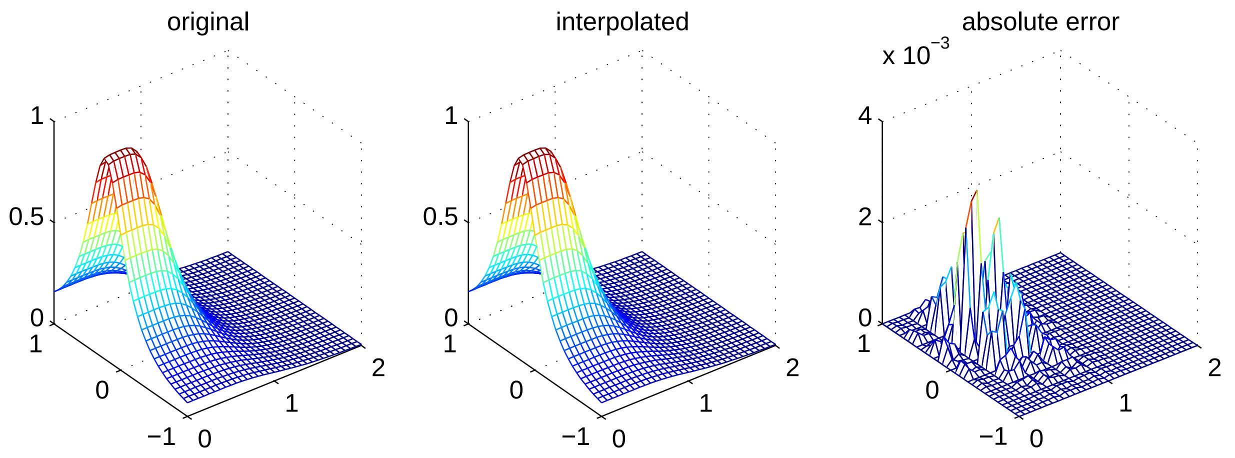

Figure 3.1: Interpolating a two-dimensional function

An ODE under uncertain input data

The second example shows how to handle multiple output arguments (in this case, multiple time-steps of an ordinary differential equation), and setting up a more complex model for the call in `spvals`.

The model considered is a second order differential equation

$$Q''(t) + aQ'(t) + b = 50\cos(t)$$

from [1, pp. 145–162] simulating an electrical circuit. Rewriting this second-order equation as a system of first order equations [4], we can define the ODE file in MATLAB format as follows:

```
function [out1, out2, out3] = ode(t, u, flag, a, b);
% ODE  definition of the electrical circuit ODE.

switch flag
 case ''
  out1 = [u(2); 50*cos(t) - a*u(2) - b*u(1)];
 case 'init'
  out1 = [0; 5];        % tspan
  out2 = [5; 1];        % initial conditions
  out3 = odeset('RelTol', 1e-6);
end
```

We can solve this ODE for $a = 2, b = 4$, and the default initial conditions and time span as defined in the ODE file using the MATLAB solver `ODE45`.

```
[t,Q] = ode45('ode',[],[],[],2,4); plot(t,Q)
```

The result is shown in Fig. 3.2.

We now consider the initial conditions and the parameters a, b to be uncertain, that is we assume intervals for $Q(0)$, $Q'(0)$, a, and b, and compute an error-controlled sparse grid interpolant for the ODE model at each time step. The interpolant can then be used to do several useful analyses, for instance, perform a Monte Carlo simulation with random variables, optimize the model for the given range of parameters and initial conditions, e.g. minimize or maximize the amplitude, or compute an envelope of the result using fuzzy calculus [2] or interval analysis. In

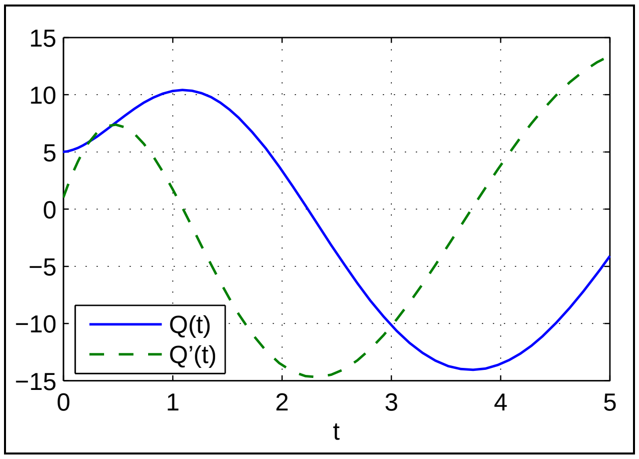

Figure 3.2: Solution of the electrical circuit system

many cases, this can be done considerably faster than by using the original ODE directly, since the construction and evaluation of the interpolant is very fast.

We proceed as follows. First of all, we write a short wrapper function of the ODE model to enable its evaluation by the `spvals` function.

```
function varargout = model(Q0, Q0prime, a, b, tspan, nsteps)
% Definition of the complete model as a function of the uncertain
% input parameters.

% The time steps must be at fixed steps such that the number of
% outputs and time steps stay the same for each parameter
% variation.
t = linspace(tspan(1), tspan(2), nsteps);

% Call the ODE solver
[t, Q] = ode45('ode', t, [Q0 Q0prime], [], a, b);

% Convert result vector to parameter list. This conversion is
% necessary, since the output arguments of the objective function
% to SPVALS must all be scalar. In this case, we assume that only
% the first column (i.e. Q, not Q') is of interest and thus
% returned.
varargout = num2cell(Q(:,1)');
```

Next, we construct the interpolant, simultaneously for all time steps. Here, we use the intervals $[Q(0)] = [4,6]$, $[Q'(0)] = [0,2]$, $[a] = [1,3]$, and $[b] = [3,5]$.

```
% MAIN script file

% Problem dimension
d = 4;

% Define the time span considered
tspan = [0 5];
```

```
% Define the number of steps to consider
nsteps = 101;

% Define the objective range of the initial conditions and the
% parameters
range = [4 6;  % [Q(0)]
         0 2;  % [Q'(0)]
         1 3;  % [a]
         3 5]; % [b]

% Maximum number of sparse grid levels to compute
nmax = 3;

% Initialize z
z = [];

% Turn insufficient depth warning off, since it is anticipated.
warning('off', 'MATLAB:spinterp:insufficientDepth');

% Compute increasingly accurate interpolants; use previous results;
% display estimated maximum relative error over all time steps at
% each iteration.
for n = 1:nmax
  options = spset('Vectorized', 'off', 'MinDepth', n, 'MaxDepth', ...
                  n, 'NumberOfOutputs', nsteps, 'PrevResults', z);
  z = spvals('model', d, range, options, tspan, nsteps);
  disp(['Current␣(estimated)␣maximum␣relative␣error␣over␣all␣time' ...
        'steps:␣', num2str(z.estRelError)]);
end

% Turn insufficient depth warning back on
warning('on', 'MATLAB:spinterp:insufficientDepth');
```

We can now compute interpolated values at each time step, for any combination of parameters within the range that the interpolant was computed for. The structure z contains all the required information. We only need to select the desired output parameter (i.e. the time step in this example). To compute 10 randomly distributed values at time $t = 5$ (which is step #101 with the chosen discretization) within the box $[Q(0)] \times [Q'(0)] \times [a] \times [b]$, we would simply use the following commands:

```
% Compute 10 randomly distributed points in [0,1] and re-scale them to
% the objective range
x = cell(1,4);
for k = 1:d
  x{k} = range(k,1) + rand(1,10) .* (range(k,2) - range(k,1));
end

% Select output parameter #101
z.selectOutput = 101;
% Compute and display interpolated values
y = spinterp(z, x{:})
```

Bibliography

[1] J. J. Buckley, E. Eslami, and T. Feuring. *Fuzzy Mathematics in Economics and Engineering*. Physica-Verlag, Heidelberg, Germany, 2002. 10

[2] A. Klimke and B. Wohlmuth. Computing expensive multivariate functions of fuzzy numbers using sparse grids. *Fuzzy Sets and Systems*, page (in press), 2005. 10

[3] A. Klimke and B. Wohlmuth. Piecewise multilinear hierarchical sparse grid interpolation in matlab. *ACM Transactions on Mathematical Software*, 2005. 4

[4] C. B. Moler. *Numerical computing with MATLAB*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2004. 10

[5] L. F. Shampine and M. W. Reichelt. The MATLAB ODE suite. *SIAM J. Sci. Comput.*, 18(1):1–22, 1997. 4