Surfe’s Inputs and Outputs

Definitions

*Interface*: a boundary surface that separates two different regions. E.g. interface that separates two lithologies, a fault surface that separates two continuous domains/volumes

*CPD function*: Conditionally Positive Definite function. Conditional in the sense the interpolation matrix is positive definite subject to the orthogonality constraints provided by the polynomial in the null space.

*SPD function*: Strictly Positive Definite function. A Gramm matrix (e.g. our interpolation matrices) generated by such a function/kernel will have all of items eigenvalues positive.

All input parameters and constraints as well as outputs to the Surfe interpolant are made through the Surfe\_API class.

Input Parameters

***Modelling method***

The modelling method parameter initializes the Surfe\_API object.

Set by Surfe\_API(int modelling\_method)

Usage example:

Surfe\_API surfe(1); // Single surface

Surfe\_API \*surfe = new Surfe\_API(2); // Lajaunie method

Acceptable integer values :

* 1: Single\_surface
  + Modelling of a single interface surface. No increment points are used in this method therefore it is a faster to obtain the interpolant and evaluate it at a series of points
* 2: Lajaunie\_approach
  + Modelling of multiple conformal interface surfaces. Increment points are used in this method.
* 3: Vector\_field
  + Modelling of vectors fields from gradient and tangent constraints. Does not incorporate interface or increment constraints.
* 4: Stratigraphic\_horizons
  + Modelling of multiple conformal interface surfaces with additional constraints on the order in which the layers/interfaces where created. E.g. Layer A is on top/younger of Layer B, Layer B is on top/younger of Layer C. This method is very useful in situations where the data does not sample the volume optimally. E.g. Interface data obtain from outcrops (horizon sampling bias) – Lajaunie method fail in then situation: The increment constraint only indicated that points on the same interface have the same scalar value – nothing about the order of these interfaces are imposed by such constraints.
* 5: Continuous\_property
  + Modelling for scalar data like assay data, points sampling a function like a fourier series. Does not incorporate gradient constraints.

***RBF kernel***

The type of RBF/Kernel used in the interpolant.

Note: If CPD kernels are used special care must be taken when using inequality constraints. This is so because in this situation, a convex optimization problem must be solved. As such these methods require a SPD matrices. To make our matrices PD we use Lagrangian polynomial basis to ensure our functional space resides within a Reproducing Hilbert Kernel Space. Currently, the only CPD function that is support those these cases is the cubic.

Set by surfe.SetRBFKernel(character array)

Acceptable strings:

* “r3”: (CPD)
* “Gaussian”: (SPD)
* “Multiquadrics »: (CPD)
* "Inverse Multiquadric" : (SPD)
* “Thin Plate Spline” : (CPD)
* “r”: (CPD)
* “WendlandC2”: (SPD)
* “MaternC4”: (SPD)

**Important**: For SPD kernels, a shape parameter is also needed. This is set by:

surfe.SetRBFShapeParameter(float)

***Polynomial order***

Polynomials of the correct minimum order are required for CPD kernels to ensure an unique minimum norm interpolant. For cubic kernels the minimum order is 1. These polynomials are the drift functions in co-kriging.

Set by surfe.SetPolynomialOrder(int)

Acceptable integer values:

* + 0 order:
  + 1 order:
  + 2 order:

***Regression Smoothing***

Regression smoothing creates an approximate interpolate instead of exact fitting. Exact fitting with real (non-synthetic data) will likely produce topological errors especially with noisy or highly varying data. If a threshold for fitting data constraints is given effectively a least squares minimization is applied to residuals. In the co-kriging world this is the nugget effect.

Set by surfe.SetRegressionSmoothing(bool, smoothing\_amount)

Note: Depending on the kernel used and the structural complexity sampled by the data, the value specified does not necessarily correspond to physical interpretable meaning: e.g. distance

***Greedy Algorithm***

Reduces the number of data/centers used by the interpolant while also accurately approximating the interpolant. Effectively this is a massive speed boost in evaluation of the interpolant. This algorithm will start with the minimum number of constraints needed to obtain an interpolant, then the interpolant will be evaluated everywhere data constraints were not included into the interpolant to measure residuals (how much discrepancy there is). A small number of residuals beyond the user specified residual threshold will then be added to the interpolant until all residuals are below this threshold. A threshold for interface data and orientation data are required.

Set by surfe.SetGreedyAlgorithm(bool, interface\_uncertaintly, angular\_uncertainty)

Note: The disadvantage of this method is that a bias is placed on outliers. Also note, that support for this method is very limited. I believe only single surface method is support.

Interface residual at point not included in interpolant: where is the scalar field constraint for the interface point. For single surface methods, this is set to 0.

Orientation residual at point not included in interpolant:

***Restricted Range***

This option is somewhat like the greedy algorithm but does not suffer from the bias on outliers. All data constraints are included into the interpolant, and every data constraint has a bound on it where the smoothest interpolant is to be constrained within. To obtain such an interpolant a quadratic optimization problem is solved, which introduces computation overhead. But only finding the interpolant compute time is affected, not the evaluation of the interpolant at user specified points.

For interface points:

For orientation points:

Set by surfe.RestrictedRange(bool, interface\_uncertaintly, angular\_uncertainty)

***Global Anisotropy***

This option derives principle directions of anisotropy using all the supplied planar/gradient constraints. These directions are used to modified how distances are computed. Works well if modelled structures are very global. However, works very poorly when plunge of structures changes a lot.

Set by surfe.SetGlobalAnisotropy(bool)

Input Constraints

There are 4 types on data constraints that can be supplied into surfe:

1. Interface constraints
   * 3D points sampling an interface that separates two volumetric domains (e.g. lithologies, fault domains)
   * (x,y,z,level)
     + Level is a float value that organizes the stratigraphic order of the interfaces and lithologies (via inequalities). Important: Larger level values are younger (on top) than smaller level values (on bottom). Points sampling the same interface must have the same level value.
     + Note: If only one interface is being model, any level value can be chosen. Normally 0 is specified in this case.
   * Added by surfe.AddInterfaceConstraint(x,y,z,level)
   * Added by surfe.SetInterfaceConstraints(Array/Matrix)
     + Nx4 matrix, n = number of interface points
     + Columns: x, y, z, level
2. Planar constraints
   * 3D points that have normal (Younging) information attributed to them. The normal indicates the direction in which younger stratigraphy is found. If fault surfaces are being modelled, the polarity of the normal does not matter.
   * There are four ways planar constraints can be added:
     + Supply the normal vector
       - Added by surfe.AddPlanarConstraint(x,y,z,nx,ny,nz)
         * nx,ny,nz are the x,y,z components of the normal vector
     + Supply strike, dip, polarity
       - Note polarity has two acceptable values: 0 == upright, 1 == overturned (points down)
       - Added by surfe.AddPlanarConstraint(x,y,z,strike, dip, polarity)
     + Supply azimuth, dip, polarity
       - Note polarity has two acceptable values: 0 == upright, 1 == overturned (points down)
       - Note that azimuth == dip direction
       - Added by surfe.AddPlanarConstraint(x,y,z,azimuth, dip, polarity)
     + Supply a matrix or array
       - Added by surfe.SetPlanarConstraints(array/matrix)
         * Nx6 matrix, N = number of constraints
         * Columns: x, y, z, nx, ny, nz
3. Tangent constraints
   * 3D points that have vector attributed to them. The relationship between this vector and the scalar field is . In other words, the vector is orthogonal (90) with respect to the gradient of the scalar field . This constraint does not have a lot of effect on changing the modelled geometry since there is a large amount of freedom on fitting this constraint. However, if a lot of these are specified in addition with supplying two tangent constraints at the same point (different vectors) then this constraint can be useful especially for foliation orientations (there is no polarity)
   * Added by surfe.AddTangentConstraint(x,y,z,tx,ty,tz)
   * Added by surfe.SetTangentConstraints(array/matrix)
     + Nx6 matrix, N = number of constraints
     + Columns: x, y, z, tx, ty, tz
4. Inequality constraints
   * 3D points sampling lithologies or inside/outside points relative to an interface/s. E.g. Do not belong to an interface. Very useful constraint since observing interface between lithologies is extremely rare and this is the most abundant type of data especially for Geological surveys.
   * (x, y, z, level)
     + Note: the level property has to be compatible and make sense with respect to the level property of the interface level property. For example, you can’t have an inequality point having the same level value as an interface point.
     + Added by surfe.AddInequalityConstraint(x,y,z,level)
     + Added by surfe.SetInequalityConstraints(Array/Matrix)
       - Nx4 matrix, n = number of interface points
       - Columns: x, y, z, level

Outputs

***Scalar field***

Obtaining the value of the scalar field at a 3D point

Get by surfe.EvaluateInterpolantAtPoint(x,y,z) – returns a double

***Gradient field***

Obtaining the gradient of the scalar fied at a 3D point

Get by surfe.EvaluateVectorInterpolantAtPoint(x,y,z) – returns a vector/array

***Spatial Parameters***

Obtain spatial metrics of the inputted data constraints

Get by surfe.GetDataBoundsAndResolution() – returns a data structure called SpatialParameters

SpatialParameters data members include (xmin, xmax, ymin, ymax, zmin, zmax, resolution)

Note resolution is the spatial resolution required to model the structural variability.

***Interface Reference Points***

For every interface being model there is a reference point for that interface (used for increments). Getting these reference points are critical when increments are used to model multiple conformal surfaces. This is because we don’t know what the scalar field values are associated to the interfaces. These values are need when applying marching cubes algorithms to extract the iso surface from the modelled scalar field. The get these values you need to evaluate the interpolant at these reference points.

Get by surfe.GetInterfaceReferencePoints() – returns an array/matrix

* Nx3 matrix, N= number of interfaces

***Number of Interfaces***

Gets the number of modelled interfaces by the interpolant.

Get by surfe.GetNumberOfInterfaces() – returns an integer

***Has Interpolant been computed***

This is a convenience method to determine whether or not the interpolant has been computed yet.

Get by surfe.InterpolantComputed() – returns a boolean