Requirements Page for DM-1991: Refactor Approximate /Interpolate

Target release	Summer 2015
Epic	⚠ DM-1991 - Jira project doesn't exist or you don't have permission to view it.
Document status	DRAFT
Document owner	Yusra AlSayyad
Assignee	Yusra AlSayyad

- Goals
- Questions
- Assumptions
- Requirements
- User Interaction
- Design
- Questions
- Not Doing

The original description of the problem is here:

⚠ DM-740 - Jira project doesn't exist or you don't have permission to view it.

The HSC implementation is simple enough that I don't see many modifications to the design needed to fit with LSST.

However, before finalizing a design and making a request for comments, I'd like to make sure I fully understand the scope and requirements. This interface will be used by many components we haven't written yet, and I would appreciate help completing this list of possible clients.

Goals

Design an abstract interface for 2D surface-modeling. Refactor Approximate/Interpolate classes to inherit from a single interface so that they can be used interchangeably, regardless of internal representation of parameters.

Questions

Please take a look at the following lists to see if there is anything I haven't captured.

- List of **client code** in the stack:
    - Current:
        - `lsst.pipe.tasks.MatchBackgrounds`
        - `afw.math.BackgroundMI`
        - `afw.math.Background`
    - Future:
        - Aperture Corrections
        - Zeropoint Scaling: Zeropoints vary spatially over a focal plane. We want a way to fit and store a model of the spatially varying zeropoint, along with the Calib.
        - Interpolate PSF across the focal plane
    - Notes: Currently the only implementations are Chebyshev polynomials, Splines which operate on gridded input data, and Gaussian Processes that operate on scattered data.

- **Domain terminology.** Sharing a consistent terminology will simplify the design process. Ideas for describing these concepts:
    - General concept of a fit 2d surface that will inspire the name of the abstract base class:

        - *Surface*?
        - *2D Model*?
        - *Bounded Field?* <-- from HSC
    - Positions of input points (two types):
        - *gridded* vs. *scattered*
    - Noise handling. How do we want to describe the difference between polynomial fitting vs. interpolation through the exact values. Assumption is that a smoothed approximation would be twice differentiable.
        - *smoothed* vs. *exact*
            - Smoothed examples:
                - Chebyshev polynomial, bicubic spline, kriging/gaussian processes, radial basis functions
            - Exact examples:
                - nearest neighbor, linear interpolation (residuals = 0, parameters are original input points)

- What basic operations do we expect to perform on these 2D Models:
    - transformations
        - Affine
        - Scale
        - Rotation may be too specific. It is difficult on gridded interpolation for example.
    - Operations on images: (image +/-/*/+/ surface)
    - Operations with other surfaces (surface = surface + another surface)
    - fillImage(), evaluate(), fit(), getResiduals()

- Expected inputs:
    - Vectors or ndarrays of x1, x2, y, weights
    - Image
    - Masked Image

Assumptions

Requirements

#	Title	User Story	Importance	Notes
1	Persistence	Aperture correction needs to save surface fits	Must Have	- ⚠ DM-832 - Jira project doesn't exist or you don't have permission to view it.
2	Gridded and Scattered input	Should use faster algorithms when input is gridded. Interface should make it easy to get the right algorithm		
3	2D-Model objects need same interface	Client code (background-matching task for example) will instantiate a 2D-Model object (whether polynomial or spline subclass will depend on the configuration - begs for a Factory). It will then call the same methods on it regardless of type.	Must have	

4	···			

User Interaction

I would like consistency with the way that the similar objects are created and used in the lsst.afw.math. For example, many require the creation of a Control which gets passed to the constructor:

```
statsCtrl = afwMath.StatisticsControl()
statsCtrl.setNumSigmaClip(self.config.sigmaClip)
statsCtrl.setNumIter(self.config.clipIter)
statsCtrl.setAndMask(self.getBadPixelMask())
statsCtrl.setNanSafe(True)
statObj = afwMath.makeStatistics(maskedImage.getVariance(), maskedImage.getMask(),
afwMath.MEANCLIP, statsCtrl)
```

I would also like consistency with APIs that other 2D-modelling code that astronomer users might be familiar with:

```
#Astropy:
from astropy.modeling import models, fitting
polynomialModel2D = models.Polynomial2D(degree=2)
fitter = fitting.LinearLSQFitter()
polynomial2D  = fitter(polynomialModel2D, x, y, z)
zNew = polynomial2D(xNew, yNew) #to evaluate

#Numpy/scipy:
from scipy import interpolate
f = interpolate.interp2d(x, y, z, kind='cubic')
zNew = f(xNew, yNew)

#Scikit-learn (1d-example)
from sklearn import GaussianProcess
gp = GaussianProcess(corr='squared_exponential', theta0=theta0...)
gp.fit(x, y)
zNew = gp.predict(xNew)
#This create then fit is consistent throughout sklearn.
```

I like the consistency of the sci-kit learn API, but these objects are not are immutable once created (see first comment).

The prototype user interaction that was presented in RFC-58:

```
chebCtrl = lsst.afw.math.Model2DControl.makeControl('CHEBYSHEV', moreConfigs)
chebyshevModel2D = lsst.afw.math.Model2D.fit(x, y, z, bbox, chebCtrl)
chebyshevModel2D.fillImage(im)



interpCtrl = lsst.afw.math.Model2DControl.makeControl('INTERPOLATE', moreConfigs)
interpModel2D = lsst.afw.math.Model2D.fit(x, y, z, bbox, interpCtrl)
interpModel2D.fillImage(im)
```

Design

Prototype design that could would enable this type of interface:

Questions

Question	Outcome
Is this refactor a candidate for rewriting the class in python? - There has been talk of redrawing the boundary between python and C++.	

Not Doing