Homework 01 – Monte Carlo Integration with Multiquadric Interpolation

Due Wednesday, 30 January 2013, 11:59:59 P.M.

Abstract

Project code name: *alexandria*. Compute the average value of an interpolated function over an irregular, user-specified, polygon using *Monte Carlo integration*[1]. A *multiquadric interpolation*[2] of irregularly-spaced, user-specified, observation data defines the interpolated function.

1 Introduction

Consider a watershed[3] defined by an irregular polygon[4]. A number of rain gauges[5] operate in the vicinity of this watershed. Some of these rain gauges may reside inside the watershed, and some may reside outside the watershed; all of the gauges provide information about rainfall in the area. Using the data provided by the rain gauges, estimate the total precipitation over the watershed.

This particular problem comes from hydrology. All subdisciplines of civil and geological engineering offer equivalent problems. For example: estimate the total volume of concrete placed for a foundation on an uneven base using thickness measurements from drilled cores[6]; or, estimate the total contaminant mass in the top soil on a brownfield[7] site using measurements from *split-spoon* samples[8].

Although the problem specifics may vary, the underlying computations are the same. You are going to write a `MATLAB` program to solve this class of problems.

This is an **individual** assignment. You may compare answers, swap ideas, and help each other out – but every individual must make their own complete submission. Make certain that your submission is correct and complete. Remember that technical competence, numerical accuracy, neatness, organization, and clarity of presentation will all be considered in evaluating your work.

2 Problem

Compute the average interpolated value over a polygonal area.

[1] See, for example, `http://en.wikipedia.org/wiki/Monte_Carlo_integration`.

[2] See, for example, `http://en.wikipedia.org/wiki/Radial_basis_function`.

[3] See, for example, `http://en.wikipedia.org/wiki/Drainage_basin`.

[4] See, for example, `http://www.mathopenref.com/polygonirregular.html`. The interactive capabilities of this web site will allow you to explore the wide variety of classes of irregular polygons.

[5] See, for example, `http://en.wikipedia.org/wiki/Rain_gage`.

[6] See, for example, `http://www.csunitec.com/core-drills/`.

[7] See, for example, `http://en.wikipedia.org/wiki/Brownfield_land`.

[8] See, for example, `http://www.ams-samplers.com/pdfs/SplitSpoon1.pdf`, or `http://www-pub.iaea.org/MTCD/publications/PDF/te_1415_web.pdf`.

3 Application Programming Interface

3.1 Function: `alexandria`

3.1.1 Usage

```
[Favg,Area] = alexandria(D,V,R,precision)
```

3.1.2 Input Arguments

- `D` is the data matrix containing the known locations and values of the specified irregularly spaced data. `D` is an (N $\times$ 3) matrix. The first column of `D` contains the x-coordinates (e.g. eastings). The second column of `D` contains the y-coordinates (e.g. northings). The third column of `D` contains the values to be interpolated (e.g. precipitation).
- `V` is the $(M + 1 \times 2)$ matrix of polygonal vertices. These vertices must be ordered to walk the perimeter of the polygon in a counter-clockwise direction (widdershins[9], or anti-clockwise). Furthermore, the list of vertices must wrap around; that is, the last vertex must equal the first vertex. The polygon so defined must be a simple polygon[10].
- `R` is the strictly positive multiquadric smoothing parameter.
- `precision` is a strictly positive scalar argument specifying the desired precision of the estimated average.

3.1.3 Preconditions

The following preconditions must be checked on input by your routine. It any of these preconditions are false, then your routine must issues a helpful, unambiguous error message and terminate execution using the `MATLAB error` command.

a. The data matrix, `D`, must have at least 1 row and exactly 3 columns.

b. None of the known data locations may coincide.

c. The polygon matrix, `V`, must have at least 4 rows and exactly 2 columns.

d. The last vertex must equal the first vertex in the polygon matrix V.

e. The area of the specified polygon must be strictly positive.

f. The multiquadric smoothing factor, `R`, must be a strictly positive scalar value.

g. The `precision` must be a strictly positive scalar value.

Note, as stated above in the Section 3.1.2, there is are additional requirements that the polygon defined by the V matrix must be simple, and that the vertices are listed in a counter-clockwise order. These would be messy – and surprisingly difficult – tests, so you will skip it for this assignment. Nonetheless, you must make certain that all of the requirements are explicitly, and clearly, stated in your online documentation – even those that are not explicitly tested.

[9] See `http://en.wikipedia.org/wiki/Widdershins`. I did not make this word up.

[10] `http://en.wikipedia.org/wiki/Simple_polygon`

3.1.4 Output Arguments

- `Favg` is a scalar value returning the estimated average interpolated value over the given polygon.
- `Area` is a scalar value returning the area of the given polygon.

4 Deliverables

4.1 Contents

Your project deliverables must include the following three necessary components.

- A one page letter report. This letter report must be in a `.pdf` format.
- An invoice, including a detailed time and task accounting. This invoice must be in a `.pdf` format.
- The code:
  - `alexandria.m`

4.2 The Letter Report

A letter report (one page or less) is all that is necessary for the formal written documentation. The letter is to be written to your client:

```
Dr. Efi Foufoula-Georgiou
Department of Civil Engineering
University of Minnesota
500 Pillsbury Drive S.E.
Minneapolis, MN 55455
```

The letter report must include the following project-specific information:

- A paragraph describing the submission. You do not need to present the details, since your client already knows the problem that he hired you to solve.
- A paragraph (or more) describing why you believe that your code is correct.
- You need to explicitly enumerate your efforts to demonstrate the correctness of your submission. Include a bulleted-type list of tests that you ran to check your program (and/or individual components) in your transmittal email. This is REQUIRED, and the contents will be considered when grading your work.

This letter report must follow a proper business letter format.

4.3 The Invoice

On a separate page, prepare a properly formatted business invoice for your work. This includes the detailed record of the hours spent on this project, as described in the course syllabus. A table may be the most effective way to present this data. Your billing rate is $100 per hour.

4.4 The Code

It is important that your code be well organized and annotated, otherwise debugging is nearly impossible. Form and format (e.g. documentation and proper indentation) will be components of your grade on this assignment[11]. Form and format matter.

Your function must include a proper title block. See the handout "matlab_style_guidelines.pdf" for more discussion on this point. See Appendix A for an example. When I type `help alexandria` at the command line I want to see a concise but clear and complete presentation.

4.5 Electronic Submission

- Your project deliverables must be submitted digitally. A paper submittal is **NOT** required or requested, and will **NOT** be considered.
- Your report documents (letter report and invoice) **shall** be submitted as ".pdf" files. Reports submitted as Microsoft Word files (.doc or .docx) will **NOT** be considered.
- Your report (.pdf file), invoice (.pdf file), and your associated program files (.m files) must be compressed into a single ".zip" file[12] and submitted via the *Homework 01 Final Submission* activity on the course *Moodle* site.
- The zip file name must include your last name and the characters "HW01". For example, "BarnesHW01.zip".

5 Hints

- When checking the precoditions make the error messages as explicit and as clear as possible. See Appendix B.
- Use a Monte Carlo integration technique to estimate the average value over the polygon. See Appendix C.
- The `MATLAB` function `inpolygon` should be used to determine is a point is inside a polygon.
- The `MATLAB` function `polyarea` should be used to compute the area of a polygon.
- Start with the layout and design of `alexandria`. Write the main sequence of comments: what is the necessary order of activities? See Appendix D.
- How are you going to generate the internal points at which you will interpolate the function?
- Design and construct a test problem, and write the associated driver function, before writing your *alexandria* code. See Appendix E for an inadequate example.

[11] Be forewarned, if you ask for help and I cannot follow your work, I will not be able to give timely and useful advice. I will have little patience for poorly written or poorly documented code. Do not send me poorly formatted or uncommented code, and ask for help.

[12] If you do not know how to make a compressed .zip file, then learn. This is a built-in capability on your computer: see, for example, `http://support.microsoft.com/kb/306531`.

A An example title block

```
%--------------------------------------------------------------------------
% Usage: [Favg,Area] = alexandria(D,V,R,precision)
%
%   Compute the average value of an interpolated function over an irregular,
%   user-specified, polygon using Monte Carlo integration.  A multiquadric
%   interpolation of irregularly-spaced, user-specified, observation data
%   defines the interpolated function.
%
% Arguments:
%   D   (n x 3) matrix of (x,y,z) data triples.
%   V   (m x 2) maxtrix of polygon vertices in counter-clockwise order.
%       The first and last vertex must coincide.
%   R   a strictly positive multiquadric smoothing parameter.
%   precision   is a strictly positive scalar argument specifying the
%       desired precision of the estimated average.
%
% Returns:
%   Favg    a scalar value returning the estimated average interpolated
%           value over the given polygon.
%   Area    is a scalar value returning the area of the given polygon.
%
% Preconditions:
%   o The data matrix, D, must have at least 1 row and exactly 3 columns.
%   o None of the known data locations may coincide.
%   o The polygon matrix, V, must have at least 4 rows and exactly 2 columns.
%   o The last vertex must equal the first vertex in the polygon matrix V.
%   o The area of the specified polygon must be strictly positive.
%   o The multiquadric smoothing factor, R, must be a strictly positive scalar.
%   o The precision must be a strictly positive scalar.
%
% Notes:
%   o Discuss the tricks and techniques used in your code here.
%   o Include references that you used to write your code.
%   o The polygon vertices must be given in counter-clockwise order, with
%       last vertex equal to the first vertex.
%
% References:
%   o R. L. Hardy. Theory and applications of the multiquadric-biharmonic
%       method. Computers Math. Applic., 19(89):163208, 1990.
%
% Version:
%   20 January 2013
%
% Written by:
%   Dr. Randal J. Barnes
%   Department of Civil Engineering
%   University of Minnesota
%--------------------------------------------------------------------------
```

B Checking preconditions

Here are a couple of examples. The first is a doubly-nested for-loop checking for coincident data pairs. Notice the limits on the for-loop statements; we only need to check each pair once. Also note the use of the `fprintf` like formating in the `error` command. This is useful for making the error message explicit and clear.

```
for j = 1:nData-1
    for k = j+1:nData
        if D(j,1)==D(k,1) && D(j,2)==D(k,2)
            error('Known data locations %d and %d coincide.', j,k);
        end
    end
end
```

The second is a simpler test. Note the use of the "not" preceding the `isscalar` command in the test.

```
if ~isscalar(R) || R <= 0
    error('The multiquadric smoothing factor, R, must be a strictly positive scalar.');
end
```

C Monte Carlo integration

C.1 The idea

The idea behind Monte Carlo integration is remarkably simple. We generate a large number of random locations within our polygonal area. At each of these locations we compute the associated function value using multiquadric interpolation. To estimate the average value over the polygon we simply compute the arithmetic average of the interpolated values.

C.2 Implementing the precision requirement

As shown in the code fragment below, we start with an empty and add more and more points (in batches, not one at a time) until the precision requirement is met. The precision requirements is embodied in the condition on the `while` statement.

```
F = [];

% Add more and more points until the precision criterion is met.
while isempty(F) || 2*std(F)/sqrt(length(F)) > precision

    % ... stuff goes here.

    F = vertcat(F,Z);
end
```

```
% Compute the final average.
Favg = mean(F);
```

The sought after `Favg` is simply the arithmetic average of the function evaluated at a whole bunch of random points inside the polygon. As such, our lessons learned about "X bar" in CE 3102 apply. The *central limit theorem*[13] tells us that `Favg` computed using N evaluations will approximately follow a Normal Distribution with a mean equal to the true average (i.e. the average as $\mathsf{N} \to \infty$), and a standard deviation equal to the true standard deviation divided by the square root of N. Thus, the simple test in the `while` condition will be "true" until the user-specified `precision` is within plus or minus two standard devaitions. That is,

$$\boldsymbol{Pr}(\texttt{Favg} - \text{precision} < \text{true average} < \texttt{Favg} + \text{precision}) > 0.95 \tag{1}$$

C.3 Generating random points in the polygon

We will use the random number generations commands available in `MATLAB` to assists us in generating the points in the polygon. Actually, what we are going to do is generate a whole bunch of random points in the neighborhood of the polygon, and throw away any of the points that fall outside the polygon. See the code fragment below. Makes certain that you understand how this works[14]

```
Xtry = unifrnd(xmin,xmax,nSet,1);
Ytry = unifrnd(ymin,ymax,nSet,1);

I = inpolygon(Xtry,Ytry,V(:,1),V(:,2));

X = Xtry(I);
Y = Ytry(I);
Z = nan(length(X),1);
```

where, as shown in the code fragment below, the `xmin`, `xmax`, `ymin`, and `ymax` define a simple axis-aligned rectangle that circumscribes the polygon. This bit of code only needs to be done once – i.e. outside of the while loop – since the polygon does not change.

```
xmin = min( V(:,1) );
xmax = max( V(:,1) );
ymin = min( V(:,2) );
ymax = max( V(:,2) );
```

[13] See, for example, `http://en.wikipedia.org/wiki/Central_limit_theorem`.

[14] In particular, investigate the use of the boolean index vector `I` – this is a very useful technique. We will use this technique frequently.

D A possible layout for `alexandria`

```
function [Favg,Area] = alexandria(D,V,R,precision)
    % Initialize some constants.

    % Check the many required preconditions.

    % Compute the multiquadric interpolation weights.

    % Prepare to carry out the interpolation.

    % Add more and more points until the precision criterion is met.

    % Compute the final average.
end
```

E An example test function

```
function flag = test_alexandria
    tic;

    n = 50;
    X = unifrnd(-20,120,n,1);
    Y = unifrnd(-20,120,n,1);
    Z = X+Y;
    D = [X,Y,Z];

    V = [0,0; 100,0; 100,100; 0,0];

    [Favg,Area] = alexandria(D,V,1,0.1)

    flag = true;

    toc;
end
```