2017-08-17

# Circular Integration Region

R. Steven Turley
*Brigham Young University*, turley@byu.edu

# Circular Integration Region

**Version 4.3**

R. Steven Turley

August 17, 2017

## Contents


**1. Introduction** 2

**2. Quadrature Rules** 3

**3. Rough Surface** 5
- 3.1. Derivation . . . 5
- 3.2. Tests . . . 6
  - 3.2.1. Constant z . . . 6
  - 3.2.2. Flat slope . . . 6
- 3.3. Half-Sphere . . . 7

**4. Accuracy** 7
- 4.1. Square Rules . . . 7
- 4.2. Circle Rules . . . 8

**5. Singular Integrals** 9
- 5.1. Transforming from Annulus or Pie to Square . . . 10
- 5.2. Dividing Square Into Triangles . . . 11
- 5.3. Transforming to Right Triangles . . . 11
  - 5.3.1. First Triangle . . . 12
  - 5.3.2. Second Triangle . . . 13
  - 5.3.3. Third Triangle . . . 13
  - 5.3.4. Fourth Triangle . . . 14
- 5.4. Inverse Tranformation and Jacobian . . . 14
- 5.5. Alternate Transformation to Right Triangles . . . 16
  - 5.5.1. Checking Jacobian Determinant . . . 17
- 5.6. Duffy Transformation . . . 18
- 5.7. Summary . . . 19
- 5.8. Concrete Example . . . 19

5.9. Fortran Implementation . . . . . 20
5.9.1. patch.f95 . . . . . 21
5.9.2. alt_patch.f95 . . . . . 22
5.9.3. test_patch.pf . . . . . 22

**6. Application Notes** **22**
6.1. Splines . . . . . 22
6.2. Fourier Transform . . . . . 23

**A. Sample Julia Code** **23**

**B. patch.f95** **24**

**C. alt_patch.f95** **27**

**D. test_patch.pf** **32**

# 1. Introduction

For a rough mirror in 3d, we will need to integrate over a 2d surface. Given the symmetries in our problem, it seemed advisable to make the integration region circular rather than rectangular. This eliminates sharp points on the surface and makes all points on the edge equidistant from the center. Dividing a circular region into patches is advisable in order to have basis functions with compact support[1] and compact regions to handle the integrable singularities in the integrand. Rectangular or triangular patches have well-developed high-order quadrature rules for integration, but can't produce a conformal circular boundary. Patches with equally spaced values of $r$ and $\theta$ have the disadvantage of significantly difference areas near the center of the surface and near the edges of the surface. A good compromise would be a central patch which is a circle divided into thirds and then annular patches with widths equal to the radius of the central circle divided into angular sections having the same area as the central circle regions as shown in Figure 1. If $a$ is the radius of the central circle, the $n^{\text{th}}$ annular ring has an area

$$A = \pi a^2(n+1)^2 - \pi a^2 n^2 \tag{1}$$
$$= \pi a^2(2n+1)\ . \tag{2}$$

It needs to be divided into $3(2n+1)$ patches for it to have the same area as the center circle arc $\pi a^2/3$.

[1] The Nystrom method doesn't use explicit basis functions, but they are implicit in the quadrature rules. Since Gaussian-type rules are exact for polynomials up to a certain order, those polynomials can be thought of as basis functions used in the expansion of the integrand.

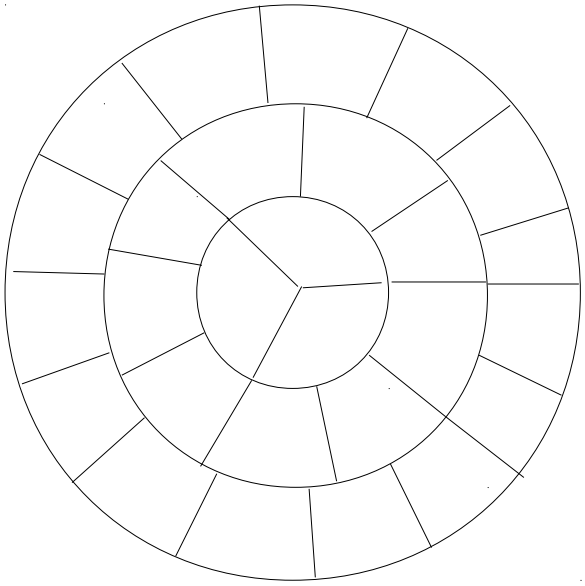

Figure 1: Annular patches with same area as center circle

## 2. Quadrature Rules

Abramowitz and Stegun have a four-point rule for integrating square regions that avoid the end points of integration[1]. The square rule is for integrating a region with

$$-h \leq x, y \leq h \; . \tag{3}$$

The rule approximates the integrals by evaluating the function at four points

$$x = \pm \frac{h}{\sqrt{3}} \tag{4}$$

$$y = \pm \frac{h}{\sqrt{3}} \; , \tag{5}$$

adding them together and multiplying the result by $h^2$. The rule has an error of order $h^4$. The square rule can be mapped onto a semi-circular annulus of inner radius $an$ and angular range

$$\frac{2\pi m}{3(2n+1)} \leq \theta \leq \frac{2\pi(m+1)}{3(2n+1)} \tag{6}$$

$$0 \leq m \leq 6n+2 \; . \tag{7}$$

The worst case should be for a center circle region with $n = 0$ and $m = 0$. Mapping the square rule onto an annulus I have the following.

$$h = \frac{a}{2} \tag{8}$$

$$r = a\left[\left(n + \frac{1}{2}\right) + \frac{y}{2h}\right] \tag{9}$$

$$= a\left(n + \frac{1}{2}\right) + y \tag{10}$$

$$= a\left(n + \frac{1}{2} \pm \frac{1}{2\sqrt{3}}\right) \tag{11}$$

$$\theta = 2\pi\frac{\left(m + \frac{1}{2}\right) + \frac{x}{2h}}{6n + 3} \tag{12}$$

$$= 2\pi\frac{a\left(m + \frac{1}{2}\right) + x}{3a(2n + 1)} \tag{13}$$

$$= 2\pi\frac{\left(m + \frac{1}{2}\right) \pm \frac{1}{2\sqrt{3}}}{3(2n + 1)} \tag{14}$$

$$\frac{\partial r}{\partial y} = 1 \tag{15}$$

$$\frac{\partial \theta}{\partial x} = \frac{2\pi}{3a(2n + 1)} \tag{16}$$

$$\int_{an}^{a(n+1)} r\,dr \int_0^{2\pi/3} f(r,\theta)\,d\theta = \frac{\pi a^2}{3} \tag{17}$$

$$\approx h^2\left(\frac{\partial r}{\partial y}\right)\left(\frac{\partial \theta}{\partial x}\right) \times \sum r\left(\pm\frac{h}{\sqrt{3}}\right) f\left(r\left(\pm\frac{h}{\sqrt{3}}\right), \theta\left(\pm\frac{h}{\sqrt{3}}\right)\right) \tag{18}$$

$$\approx \left(\frac{a}{2}\right)^2 \frac{2\pi}{3a(2n + 1)} \sum\left[a\left(n + \frac{1}{2}\right) \pm \frac{a}{2\sqrt{3}}\right] f(r,\theta) \tag{19}$$

$$\approx \frac{\pi a^2}{6(2n + 1)} \sum\left[\left(n + \frac{1}{2} \pm \frac{1}{2\sqrt{3}}\right)\right] f(r(y), \theta(x)) \tag{20}$$

Letting

$$r_+ = a\left(n + \frac{1}{2} + \frac{1}{2\sqrt{3}}\right) \tag{21}$$

$$r_- = a\left(n + \frac{1}{2} - \frac{1}{2\sqrt{3}}\right) \tag{22}$$

$$\theta_+ = 2\pi\frac{\left(m + \frac{1}{2}\right) + \frac{1}{2\sqrt{3}}}{3(2n+1)} \tag{23}$$

$$\theta_- = 2\pi\frac{\left(m + \frac{1}{2}\right) - \frac{1}{2\sqrt{3}}}{3(2n+1)} \tag{24}$$

this can be written for the entire circle as

$$\int_0^s r\,dr\int_0^{2\pi} f(r,\theta)\,d\theta \approx \frac{\pi a^2}{12}\sum_{n=0}^{N-1}\sum_{m=0}^{2n+2}\frac{2n+1+\frac{1}{\sqrt{3}}}{2n+1}\left[f(r_+,\theta_+)+f(r_+,\theta_-)\right]+ \frac{2n+1-\frac{1}{\sqrt{3}}}{2n+1}\left[f(r_-,\theta_+)+f(r_-,\theta_-)\right] \tag{25}$$

# 3. Rough Surface

The next thing to consider is what happens if the surface over which we are integrating isn't quite a circle. This could happen because the circle has a rough surface or because the area isn't a flat circle, but has some height to it. These are really the same cases, but I'll derive and test the needed metric tensor using the language of the second case.

## 3.1. Derivation

Let the surface over which I'm integrating be some height $z$ above the circle itself which we'll take to be in the x-y plane. A differential Cartesian area element will have two sides

$$\vec{S}_x = dx\,\hat{x} + \partial z_x\,\hat{z} \tag{26}$$

$$= dx\left(\hat{x} + \frac{\partial z}{\partial x}\hat{z}\right) \tag{27}$$

$$\vec{S}_y = dy\,\hat{y} + \partial z_y\,\hat{z} \tag{28}$$

$$= dy\left(\hat{y} + \frac{\partial z}{\partial y}\hat{z}\right) \tag{29}$$

where $\partial z_u$ represents the variation in $z$ keeping the coordinate $u$ constant. The surface defined by these two vectors is a parallelogram with an area

$$dA = S_x S_y \sin\theta \tag{30}$$
$$= \left|\vec{S}_x \times \vec{S}_y\right| \tag{31}$$
$$= dx\, dy \left|\hat{z} - \frac{\partial z}{\partial y}\hat{y} - \frac{\partial z}{\partial x}\hat{x}\right| \tag{32}$$
$$= dx\, dy \sqrt{1 + \left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2} \tag{33}$$

where $\theta$ is the angle between $\vec{S}_x$ and $\vec{S}_y$.

## 3.2. Tests

I checked three cases for reasonableness

### 3.2.1. Constant z

If $z$ is constant, $dA$ should be

$$dA = dx\, dy\ . \tag{34}$$

Since the partial derivatives are equal to zero, that is indeed the case.

### 3.2.2. Flat slope.

If

$$z = \alpha + \beta x \tag{35}$$

the area is a a translation of a sloped line of length

$$\ell = \sqrt{1 + \beta^2} \tag{36}$$
$$dA = dx\, dy \sqrt{1 + \beta^2}\ . \tag{37}$$

Since

$$\frac{\partial z}{\partial x} = \beta \tag{38}$$
$$\frac{\partial z}{\partial y} = 0 \tag{39}$$

this agrees with Equation 33.

### 3.3. Half-Sphere

A sphere of radius $a$ has a surface area of

$$A = \frac{4}{3}\pi a^2 \tag{40}$$

so a half sphere would have an area

$$A = \frac{2}{3}\pi a^2 \ . \tag{41}$$

Performing an integral over unit circle with a half-spherical dome above it I have

$$z = \sqrt{1 - x^2 - y^2} \tag{42}$$

$$\frac{\partial z}{\partial x} = -\frac{x}{z} \tag{43}$$

$$\frac{\partial z}{\partial y} = -\frac{y}{z} \tag{44}$$

$$\begin{aligned} dA &= dx\, dy \sqrt{1 + \frac{x^2}{z^2} + \frac{y^2}{z^2}} \\ &= z\, dx\, dy \sqrt{x^2 + y^2 + z^2} && (45) \\ &= z\, dx\, dy \sqrt{x^2 + y^2 + 1 - x^2 - y^2} && (46) \\ &= z\, dx\, dy \ . && (47) \end{aligned}$$

Switch to polar coordinates to do the integral.

$$dx\, dy = r dr\, d\theta \tag{48}$$

$$\begin{aligned} z &= \sqrt{1 - r^2 \cos^2\theta - r^2 \sin^2\theta} && (49) \\ &= \sqrt{1 - r^2} && (50) \\ A &= \int_0^1 r\sqrt{1 - r^2}\, dr \int_0^{2\pi} d\theta && (51) \\ &= 2\pi \int_0^1 r\sqrt{1 - r^2} dr && (52) \\ &= \frac{2}{3}\pi && (53) \end{aligned}$$

This is the expected answer.

## 4. Accuracy

### 4.1. Square Rules

The square rules are exact for the following monomial cases:

- any odd powers of $x$ and/or $y$ (rules give 0 since they have the appropriate symmetry
- 1
- $x^2$
- $y^2$
- $x^2y^2$

Translate these into an arc on the $n^{\text{th}}$ annulus and the $m^{\text{th}}$ section of a circle the inner radius $a$ with $-\frac{a}{2} \le x \le \frac{a}{2}$ and $-\frac{a}{2} \le y \le \frac{a}{2}$.

$$r = \left(n + \frac{1}{2}\right) a + y \tag{54}$$

$$\theta = \frac{2\pi(m + \frac{1}{2} + \frac{x}{a})}{3(2n+1)} \tag{55}$$

$$= \frac{\pi(2ma + a + 2x)}{3a(2n+1)} \tag{56}$$

This assumes $n$ and $m$ are indexed starting with 0. Since $r \propto y$ and $\theta \propto x$ these rules should be good for the same powers of $r$ and $\theta$ as they are for $x$ and $y$. Note that these values of $x$ and $y$ are the ones for the underlying square, not the actual x and y coordinates oon the arc.

## 4.2. Circle Rules

For any $n$ and $f = 1$, Equation 20 gives the exact answer of $\pi a^2/3$. For $n = 0$, $m = 0$, and $f = \cos\theta$ the exact answer is

$$\int_0^{2\pi/3} \cos\theta \, d\theta \int_0^a r \, dr = \frac{a^2}{2} \sin\theta|_{\theta=0}^{2\pi/3} \tag{57}$$

$$= \frac{a^2\sqrt{3}}{4} \tag{58}$$

$$= 0.433a^2 \, . \tag{59}$$

The numerical approximation yields

$$\int_0^{2\pi/3} \cos\theta \, d\theta \int_0^a r \, dr \approx a^2 \frac{\pi}{6} \sum \frac{\sqrt{3} \pm 1}{2\sqrt{3}} \cos\left(2\pi \frac{\frac{1}{2} \pm \frac{1}{2\sqrt{3}}}{3}\right) \tag{60}$$

$$\approx \frac{\pi a^2}{6} \sum \cos\left(\frac{\pi(\sqrt{3} \pm 1)}{3\sqrt{3}}\right) \tag{61}$$

$$\approx 0.431a^2 \tag{62}$$

which is a good approximation. For $n = 1$, $m = 0$, $f = \cos\theta$ the exact answer is

$$\int_0^{2\pi/9} \cos\theta \int_a^{2a} r\, dr = \frac{3a^2}{2} \sin\theta|_{\theta=0}^{2\pi/9} \tag{63}$$

$$= 0.9642a^2 \, . \tag{64}$$

The approximation yields

$$\int_0^{2\pi/9} \cos\theta \int_a^{2a} r\, dr \approx \frac{\pi a^2}{6} \sum \cos\left(2\pi \frac{\frac{1}{2} \pm \frac{1}{2\sqrt{3}}}{9}\right) \tag{65}$$

$$\approx \frac{\pi a^2}{6} \sum \cos\left(\frac{\pi(\sqrt{3} \pm 1)}{9\sqrt{3}}\right) \tag{66}$$

$$\approx 0.9641a^2 \tag{67}$$

which agrees to four significant digits. I did some studies with a Julia implementation and made the following observations:

- The square rules on which these rules are based are a product of Gaussian quadrature rules. On each square, calculations are exact for polynomials up to power 3 in $x$ and $y$. In other words, any of the following functions can be integrated exactly over a single square:

$$f(x, y) = x \tag{68}$$

$$f(x, y) = y \tag{69}$$

$$f(x, y) = C \tag{70}$$

$$f(x, y) = xy \tag{71}$$

$$f(x, y) = x^2 y \tag{72}$$

$$f(x, y) = x^3 y^2 \tag{73}$$

$$f(x, y) = x^3 y^3 \tag{74}$$

$$f(x, y) = (a + bx + cx^3)(d + ey^2 + gy^3) \tag{75}$$

- The integrations will likewise be exact for any power of $r$ from $-1$ to 2.
- The integrations will be exact for any power of $\theta$ from 0 to 3.
- The integrations will converge very quickly for integrations of trig functions which are periodic on a circle.

## 5. Singular Integrals

For integrals on the same patch, the Greene's function is singular. There is a series of coordinate transformations that make the integrals non-singular so that they can be integrated with Gaussian quadrature product rules. I will out line the series of transformations in this section and give some examples using Mathematica and Fortran.

## 5.1. Transforming from Annulus or Pie to Square

The first transformation needed is similar to the one used for the quadrature rule developed in previous sections. Solving Equation 13 for $x$, we have

$$x = a\left[\frac{\theta}{2\pi}(6n+3) - m - \frac{1}{2}\right] \tag{76}$$

$$\frac{\partial\theta}{\partial x} = a\frac{6n+3}{2\pi}\ . \tag{77}$$

Solving Equation 10 for $y$, we have

$$y = r - a\left(n + \frac{1}{2}\right) \tag{78}$$

$$\frac{\partial r}{\partial y} = 1 \tag{79}$$

giving us a Jacobian matrix $J$.

$$J = \left\| \begin{array}{cc} \frac{\partial r}{\partial x} & \frac{\partial r}{\partial y} \\ \frac{\partial\theta}{\partial x} & \frac{\partial\theta}{\partial\theta y} \end{array} \right\| \tag{80}$$

$$= \left\| \begin{array}{cc} 0 & 1 \\ \frac{2\pi}{a(6n+3)} & 0 \end{array} \right\| \tag{81}$$

$$= \frac{2\pi}{a(6n+3)} \tag{82}$$

The transformed integration is

$$\theta_{min} = \frac{2\pi m}{3(2n+1)} \tag{83}$$

$$\theta_{max} = \frac{2\pi(m+1)}{3(2n+1)} \tag{84}$$

$$\int_{na}^{(n+1)a} r\,dr \int_{\theta_{min}}^{\theta_{max}} f(r,\theta)\,d\theta = J\int_{-\frac{a}{2}}^{\frac{a}{2}}\left[a\left(n+\frac{1}{2}\right)+y\right]\,dy\int_{-\frac{a}{2}}^{\frac{a}{2}} f(r(y),\theta(x))\,dx\ . \tag{85}$$

To make sure the Jacobian is right, let's check this forumula for the specific case of $n = 1$, $m = 0$, $f(r,\theta) = 1$.

$$\int_a^{2a} r\,dr\int_0^{2\pi/9} d\theta = \frac{1}{2}\left[(2a)^2 - a^2\right]\frac{2\pi}{9} \tag{86}$$

$$= \frac{\pi a^2}{3} \tag{87}$$

$$\frac{2\pi}{9a}\int_{-\frac{a}{2}}^{\frac{a}{2}}\left[\frac{3}{2}a+y\right]\,dy\int_{-\frac{a}{2}}^{\frac{a}{2}} dx = \left(\frac{2\pi}{9a}\right)\left(\frac{3}{2}a^2\right)(a) \tag{88}$$

$$= \frac{\pi a^2}{3} \tag{89}$$

Note that the above formulas work for both an annulus ($n > 1$) and a pie-shape ($n = 0$).

## 5.2. Dividing Square Into Triangles

The next step is to divide the square into triangles with the singular points at a vertex of the triangle. There will be four triangles with the following verticies (listing the singular vertex first).

1. $\left(\pm a/2\sqrt{3}, \pm a/2\sqrt{3}\right)$, $(-a/2, -a/2)$, $(-a/2, a/2)$
2. $\left(\pm a/2\sqrt{3}, \pm a/2\sqrt{3}\right)$, $(-a/2, a/2)$, $(a/2, a/2)$
3. $\left(\pm a/2\sqrt{3}, \pm a/2\sqrt{3}\right)$, $(a/2, a/2)$, $(a/2, -a/2)$
4. $\left(\pm a/2\sqrt{3}, \pm a/2\sqrt{3}\right)$, $(a/2, -a/2)$, $(-a/2, -a/2)$

## 5.3. Transforming to Right Triangles

The above triangles are not right triangles. The Duffy transformation is in terms of a right triangle with

$$0 \le x \le 1 \tag{90}$$
$$0 \le y \le 1 \; . \tag{91}$$

To do this, we first translate the axes to the origin.

$$x' = x \mp \frac{a}{2\sqrt{3}} \tag{92}$$
$$y' = y \mp \frac{a}{2\sqrt{3}} \tag{93}$$

This gives us the following four triangles.

1. $(0,0)$, $(-a/2 \mp a/2\sqrt{3}, -a/2 \mp a/2\sqrt{3})$, $(-a/2 \mp a/2\sqrt{3}, a/2 \mp a/2\sqrt{3})$
2. $(0,0)$, $(-a/2 \mp a/2\sqrt{3}, a/2 \mp a/2\sqrt{3})$, $(a/2 \mp a/2\sqrt{3}, a/2 \mp a/2\sqrt{3})$
3. $(0,0)$, $(a/2 \mp a/2\sqrt{3}, a/2 \mp a/2\sqrt{3})$, $(a/2 \mp a/2\sqrt{3}, -a/2 \mp a/2\sqrt{3})$
4. $(0,0)$, $(a/2 \mp a/2\sqrt{3}, -a/2 \mp a/2\sqrt{3})$, $(-a/2 \mp a/2\sqrt{3}, -a/2 \mp a/2\sqrt{3})$

   Next, we do a linear transformation to a new coordinate system where each let of the right triangle is parallel to a coordinate axis. Let the coordinates of the second vertex of the triangle be

   $$x' = p_2 \tag{94}$$
   $$y' = q_2 \tag{95}$$

   and the coordinates of the thrid vertex be

   $$x' = p_3 \tag{96}$$
   $$y' = q_3 \; . \tag{97}$$

Then the transformation is as follows.

$$\begin{pmatrix} x'' \\ y'' \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} \tag{98}$$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} p_2 \\ q_2 \end{pmatrix} \tag{99}$$

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} p_3 \\ q_3 \end{pmatrix} \tag{100}$$

Equations 99 and 100 are four equations with the four unknowns being the elements of the transformation matrix to transform the primed coordinates to the double primed coordinates. The equation to solve for $A_{11}$ and $A_{12}$ is

$$\begin{pmatrix} p_2 & q_2 \\ p_3 & q_3 \end{pmatrix} \begin{pmatrix} A_{11} \\ A_{12} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{101}$$

and for $A_{21}$ and $A_{22}$ is

$$\begin{pmatrix} p_2 & q_2 \\ p_3 & q_3 \end{pmatrix} \begin{pmatrix} A_{21} \\ A_{22} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} . \tag{102}$$

### 5.3.1. First Triangle

The intermediate steps are kind of messy, but with the help of Mathematica, I got the following results for the first triangle.

- Singularity at $(x, y) = \left(-a/2\sqrt{3}, -a/2\sqrt{3}\right)$

$$A_1^{(1)} = \begin{pmatrix} -\frac{3+\sqrt{3}}{a} & 0 \\ -\frac{1}{a} & \frac{1}{a} \end{pmatrix} \tag{103}$$

- Singularity at $(x, y) = \left(-a/2\sqrt{3}, a/2\sqrt{3}\right)$

$$A_2^{(1)} = \begin{pmatrix} -\frac{3+\sqrt{3}}{a} & 0 \\ -\frac{2+\sqrt{3}}{a} & \frac{1}{a} \end{pmatrix} \tag{104}$$

- Singularity at $(x, y) = \left(a/2\sqrt{3}, a/2\sqrt{3}\right)$

$$A_3^{(1)} = \begin{pmatrix} \frac{-3+\sqrt{3}}{a} & 0 \\ -\frac{1}{a} & \frac{1}{a} \end{pmatrix} \tag{105}$$

- Singularity at $(x, y) = \left(a/2\sqrt{3}, -a/2\sqrt{3}\right)$

$$A_4^{(1)} = \begin{pmatrix} \frac{-3+\sqrt{3}}{a} & 0 \\ \frac{-2+\sqrt{3}}{a} & \frac{1}{a} \end{pmatrix} \tag{106}$$

### 5.3.2. Second Triangle

I got the following results for the second triangle.

- Singularity at $(x, y) = \left(-a/2\sqrt{3}, -a/2\sqrt{3}\right)$

$$A_1^{(2)} = \begin{pmatrix} 0 & \frac{3-\sqrt{3}}{a} \\ \frac{1}{a} & \frac{2-\sqrt{3}}{a} \end{pmatrix} \tag{107}$$

- Singularity at $(x, y) = \left(-a/2\sqrt{3}, a/2\sqrt{3}\right)$

$$A_2^{(2)} = \begin{pmatrix} 0 & \frac{3+\sqrt{3}}{a} \\ \frac{1}{a} & \frac{1}{a} \end{pmatrix} \tag{108}$$

- Singularity at $(x, y) = \left(a/2\sqrt{3}, a/2\sqrt{3}\right)$

$$A_3^{(2)} = \begin{pmatrix} 0 & \frac{3+\sqrt{3}}{a} \\ \frac{1}{a} & \frac{2+\sqrt{3}}{a} \end{pmatrix} \tag{109}$$

- Singularity at $(x, y) = \left(a/2\sqrt{3}, -a/2\sqrt{3}\right)$

$$A_4^{(2)} = \begin{pmatrix} 0 & \frac{3-\sqrt{3}}{a} \\ \frac{1}{a} & \frac{1}{a} \end{pmatrix} \tag{110}$$

### 5.3.3. Third Triangle

I got the following results for the third triangle.

- Singularity at $(x, y) = \left(-a/2\sqrt{3}, -a/2\sqrt{3}\right)$

$$A_1^{(3)} = \begin{pmatrix} \frac{3-\sqrt{3}}{a} & 0 \\ \frac{1}{a} & -\frac{1}{a} \end{pmatrix} \tag{111}$$

- Singularity at $(x, y) = \left(-a/2\sqrt{3}, a/2\sqrt{3}\right)$

$$A_2^{(3)} = \begin{pmatrix} \frac{3-\sqrt{3}}{a} & 0 \\ \frac{2-\sqrt{3}}{a} & -\frac{1}{a} \end{pmatrix} \tag{112}$$

- Singularity at $(x, y) = \left(a/2\sqrt{3}, a/2\sqrt{3}\right)$

$$A_3^{(3)} = \begin{pmatrix} \frac{3+\sqrt{3}}{a} & 0 \\ \frac{1}{a} & -\frac{1}{a} \end{pmatrix} \tag{113}$$

- Singularity at $(x, y) = \left(a/2\sqrt{3}, -a/2\sqrt{3}\right)$

$$A_4^{(3)} = \begin{pmatrix} \frac{3+\sqrt{3}}{a} & 0 \\ \frac{2+\sqrt{3}}{a} & -\frac{1}{a} \end{pmatrix} \tag{114}$$

### 5.3.4. Fourth Triangle

I got the following results for the fourth triangle.

- Singularity at $(x, y) = \left(-a/2\sqrt{3}, -a/2\sqrt{3}\right)$

$$A_1^{(4)} = \begin{pmatrix} 0 & -\frac{3+\sqrt{3}}{a} \\ -\frac{1}{a} & -\frac{2+\sqrt{3}}{a} \end{pmatrix} \tag{115}$$

- Singularity at $(x, y) = \left(-a/2\sqrt{3}, a/2\sqrt{3}\right)$

$$A_2^{(4)} = \begin{pmatrix} 0 & \frac{-3+\sqrt{3}}{a} \\ -\frac{1}{a} & -\frac{1}{a} \end{pmatrix} \tag{116}$$

- Singularity at $(x, y) = \left(a/2\sqrt{3}, a/2\sqrt{3}\right)$

$$A_3^{(4)} = \begin{pmatrix} 0 & \frac{-3+\sqrt{3}}{a} \\ -\frac{1}{a} & \frac{-2+\sqrt{3}}{a} \end{pmatrix} \tag{117}$$

- Singularity at $(x, y) = \left(a/2\sqrt{3}, -a/2\sqrt{3}\right)$

$$A_4^{(4)} = \begin{pmatrix} 0 & -\frac{3+\sqrt{3}}{a} \\ -\frac{1}{a} & -\frac{1}{a} \end{pmatrix} \tag{118}$$

## 5.4. Inverse Tranformation and Jacobian

To practically apply these transforms, on needs to go from $x''$ to $x$, requiring the inverse of the $A$ matrices which I'll call $B$. For completeness, I've included their explicit forms

here.

$$B_1^{(1)} = \begin{pmatrix} \frac{1}{6}\left(-3+\sqrt{3}\right)a & 0 \\ \frac{1}{6}\left(-3+\sqrt{3}\right)a & a \end{pmatrix} \tag{119}$$

$$B_2^{(1)} = \begin{pmatrix} \frac{1}{6}\left(-3+\sqrt{3}\right)a & 0 \\ -\frac{1}{6}\left(3+\sqrt{3}\right)a & a \end{pmatrix} \tag{120}$$

$$B_3^{(1)} = \begin{pmatrix} \frac{1}{6}\left(-3-\sqrt{3}\right)a & 0 \\ \frac{1}{6}\left(-3-\sqrt{3}\right)a & a \end{pmatrix} \tag{121}$$

$$B_4^{(1)} = \begin{pmatrix} \frac{1}{6}\left(-3-\sqrt{3}\right)a & 0 \\ \frac{1}{6}\left(-3+\sqrt{3}\right)a & a \end{pmatrix} \tag{122}$$

$$B_1^{(2)} = \begin{pmatrix} \frac{1}{6}\left(-3+\sqrt{3}\right)a & a \\ \frac{1}{6}\left(3+\sqrt{3}\right)a & 0 \end{pmatrix} \tag{123}$$

$$B_2^{(2)} = \begin{pmatrix} \frac{1}{6}\left(-3+\sqrt{3}\right)a & a \\ \frac{1}{6}\left(3-\sqrt{3}\right)a & 0 \end{pmatrix} \tag{124}$$

$$B_3^{(2)} = \begin{pmatrix} -\frac{1}{6}\left(3+\sqrt{3}\right)a & a \\ \frac{1}{6}\left(3-\sqrt{3}\right)a & 0 \end{pmatrix} \tag{125}$$

$$B_4^{(2)} = \begin{pmatrix} \frac{1}{6}\left(-3-\sqrt{3}\right)a & a \\ \frac{1}{6}\left(3+\sqrt{3}\right)a & 0 \end{pmatrix} \tag{126}$$

$$B_1^{(3)} = \begin{pmatrix} \frac{1}{6}\left(3+\sqrt{3}\right)a & 0 \\ \frac{1}{6}\left(3+\sqrt{3}\right)a & -a \end{pmatrix} \tag{127}$$

$$B_2^{(3)} = \begin{pmatrix} \frac{1}{6}\left(3+\sqrt{3}\right)a & 0 \\ -\frac{1}{6}\left(-3+\sqrt{3}\right)a & -a \end{pmatrix} \tag{128}$$

$$B_3^{(3)} = \begin{pmatrix} \frac{1}{6}\left(3-\sqrt{3}\right)a & 0 \\ \frac{1}{6}\left(3-\sqrt{3}\right)a & -a \end{pmatrix} \tag{129}$$

$$B_4^{(3)} = \begin{pmatrix} \frac{1}{6}\left(3-\sqrt{3}\right)a & 0 \\ \frac{1}{6}\left(3+\sqrt{3}\right)a & -a \end{pmatrix} \tag{130}$$

$$B_1^{(4)} = \begin{pmatrix} \frac{1}{6}\left(3+\sqrt{3}\right)a & -a \\ \frac{1}{6}\left(-3+\sqrt{3}\right)a & 0 \end{pmatrix} \tag{131}$$

$$B_2^{(4)} = \begin{pmatrix} \frac{1}{6}\left(3+\sqrt{3}\right)a & -a \\ \frac{1}{6}\left(-3-\sqrt{3}\right)a & 0 \end{pmatrix} \tag{132}$$

$$B_3^{(4)} = \begin{pmatrix} \frac{1}{6}\left(3-\sqrt{3}\right)a & -a \\ \frac{1}{6}\left(-3-\sqrt{3}\right)a & 0 \end{pmatrix} \tag{133}$$

$$B_4^{(4)} = \begin{pmatrix} \frac{1}{6}\left(3-\sqrt{3}\right)a & -a \\ \frac{1}{6}\left(-3+\sqrt{3}\right)a & 0 \end{pmatrix} \tag{134}$$

Here are the absolute values of the associated Jacobian determinants

$$J_1^{(1)} = \frac{(3-\sqrt{3})a^2}{6} \tag{135}$$
$$J_2^{(1)} = \frac{(3-\sqrt{3})a^2}{6} \tag{136}$$
$$J_3^{(1)} = \frac{(3+\sqrt{3})a^2}{6} \tag{137}$$
$$J_4^{(1)} = \frac{(3+\sqrt{3})a^2}{6} \tag{138}$$
$$J_1^{(2)} = \frac{(3+\sqrt{3})a^2}{6} \tag{139}$$
$$J_2^{(2)} = \frac{(3-\sqrt{3})a^2}{6} \tag{140}$$
$$J_3^{(2)} = \frac{(3-\sqrt{3})a^2}{6} \tag{141}$$
$$J_4^{(2)} = \frac{(3+\sqrt{3})a^2}{6} \tag{142}$$
$$J_1^{(3)} = \frac{(3+\sqrt{3})a^2}{6} \tag{143}$$
$$J_2^{(3)} = \frac{(3+\sqrt{3})a^2}{6} \tag{144}$$
$$J_3^{(3)} = \frac{(3-\sqrt{3})a^2}{6} \tag{145}$$
$$J_4^{(3)} = \frac{(3-\sqrt{3})a^2}{6} \tag{146}$$
$$J_1^{(4)} = \frac{(3-\sqrt{3})a^2}{6} \tag{147}$$
$$J_2^{(4)} = \frac{(3+\sqrt{3})a^2}{6} \tag{148}$$
$$J_3^{(4)} = \frac{(3+\sqrt{3})a^2}{6} \tag{149}$$
$$J_4^{(4)} = \frac{(3-\sqrt{3})a^2}{6} \tag{150}$$

## 5.5. Alternate Transformation to Right Triangles

We really don't need the $A$ matrices in our calculations, just the $B$ matrices and the Jacobian. These can be calculated in a more straightforward way. This will also serve as a check of the above calculations.

As we do these transformations, it also makes sense to number the singular points differently. If we number the points using same order as the closest second vertex of the four triangles from Section 5.2, the equations will be more symmetric.

The inverse transformations can be found directly from Equations 98 through 100.

$$\begin{pmatrix} B_{11}B_{12} \\ B_{21}B_{22} \end{pmatrix} \begin{pmatrix} x'' \\ y'' \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix} \tag{151}$$

$$\begin{pmatrix} B_{11}B_{12} \\ B_{21}B_{22} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} p_2 \\ q_2 \end{pmatrix} \tag{152}$$

$$\begin{pmatrix} B_{11}B_{12} \\ B_{21}B_{22} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} p_3 \\ q_3 \end{pmatrix} \tag{153}$$

These equations can be solved simply for $B$ in terms of the known variables $p$ and $q$ for the vertices. Mulitplying our the matrices in Equations 152 and 153 explicitly gives us the following four equations.

$$B_{11} = p_2 \tag{154}$$

$$B_{21} = q_2 \tag{155}$$

$$B_{11} + B_{12} = p_3 \tag{156}$$

$$B_{21} + B_{22} = q_3 \tag{157}$$

These are straighforward to solve for $B$.

$$B = \begin{pmatrix} p_2 & p_3 - p_2 \\ q_2 & q_3 - q_2 \end{pmatrix} \tag{158}$$

Using the vertices from Section 5.2, we have get the same values for $B$ and $J$ as in Section 5.4.

### 5.5.1. Checking Jacobian Determinant

I'll check the Jacobian determinant by integrating a unit function as before assuming a (non-existant) singularity at $(x, y) = \left(a/2\sqrt{3}, a/2\sqrt{3}\right)$. The square has an area of $a^2$ before being subdivided. The translation of the center doesn't change the area. The four triangles after transformation have areas of

$$A_t = \frac{J_i}{2} \, . \tag{159}$$

The four triangles in this case have Jacobian respective determinants of

$$J_4^{(1)} = \frac{(3+\sqrt{3})a^2}{6} \tag{160}$$

$$J_4^{(2)} = \frac{(3-\sqrt{3})a^2}{6} \tag{161}$$

$$J_4^{(3)} = \frac{(3-\sqrt{3})a^2}{6} \tag{162}$$

$$J_4^{(4)} = \frac{(3+\sqrt{3})a^2}{6} \tag{163}$$

whose sum is

$$J_4^{(1)} + J_4^{(2)} + J_4^{(3)} + J_4^{(4)} = 2a^2 \; . \tag{164}$$

This gives a total area of $a^2$ which agrees with the untransformed square.

## 5.6. Duffy Transformation

The next step is to turn each of the right triangles into squares. The $x$ coordinate will transform directly into a new $u$ coordinate. The new $v$ coordinate will be $y''/x''$. The old domains for the triangles were

$$0 \le x'' \le 1 \tag{165}$$

$$0 \le y'' \le x'' \; . \tag{166}$$

With the transformation

$$x'' = u \tag{167}$$

$$y'' = uv \tag{168}$$

$$u = x'' \tag{169}$$

$$v = \frac{y''}{x''} \tag{170}$$

the Jacobian determinant is

$$J = \left\| \begin{array}{cc} \frac{\partial x''}{\partial u} & \frac{\partial x''}{\partial v} \\ \frac{\partial y''}{\partial u} & \frac{\partial y''}{\partial v} \end{array} \right\| \tag{171}$$

$$= \left| \begin{array}{cc} 1 & 0 \\ v & u \end{array} \right| \tag{172}$$

$$= u \; . \tag{173}$$

Check this transformation with integrating a function $f(x'', y'') = 1$.

$$\int_0^1 dx'' \int_0^{x''} dy'' = \int_0^1 x''\, dx'' \tag{174}$$

$$= \frac{1}{2} \tag{175}$$

$$\int_0^1 u\, du \int_0^1 dv = \int_0^1 u\, du \tag{176}$$

$$= \frac{1}{2} \tag{177}$$

## 5.7. Summary

To integrate $rf(r,\theta)$ over the arc with $n$ specifying the arc number (starting with 0) and $m$ specifying the segment within the arc (starting at 0) use the following substitutions.

$$r = y + a\left(n + \frac{1}{2}\right) \tag{178}$$

$$\theta = 2\pi\frac{a\left(m+\frac{1}{2}\right)+x}{3a(2n+1)} \tag{179}$$

$$x = x' \pm \frac{a}{2\sqrt{3}} \tag{180}$$

$$y = y' \pm \frac{a}{2\sqrt{3}} \tag{181}$$

$$x' = B_{11}x'' + B_{12}y'' \tag{182}$$

$$y' = B_{21}x'' + B_{22}y'' \tag{183}$$

$$x'' = u \tag{184}$$

$$y'' = uv \tag{185}$$

Following down the chain, this let's you compute $r(u,v)$ and $\theta(u,v)$. Then you just substitue, multiply by the Jacobians, and integrate.

$$\int rf(r,\theta)\,dr\,d\theta = \left[\frac{2\pi}{a(6n+3)}\right] J_i^{(j)} \int_0^1 u\,du \int_0^1 r(u,v)f(r(u,v),\theta(u,v))\,dv \tag{186}$$

## 5.8. Concrete Example

This is an example of how to implement these formulas in a specific case. I will consider the case where the singularity is at $(x,y) = (a/2\sqrt{3}, a/2\sqrt{3}$. To avoid too many complication, I'll let $f(r,\theta) = G(\rho)$, ignoring any roughness on the surface. I'll define

$$G(\rho) = \frac{e^{ik\rho}}{4\pi\rho} \tag{187}$$

$$\rho = \sqrt{(r\cos\theta - r'\cos\theta')^2 + (r\sin\theta - r'\sin\theta')^2} \tag{188}$$

where

$$(r',\theta') = \left(a\left[\frac{1}{2\sqrt{3}} + n + \frac{1}{2}\right], 2\pi\frac{m+\frac{1}{2}+\frac{1}{2\sqrt{3}}}{6n+3}\right) \tag{189}$$

$$= \left(a\frac{2\sqrt{3}n+\sqrt{3}+1}{2\sqrt{3}}, 2\pi\frac{2\sqrt{3}m+\sqrt{3}+1}{6\sqrt{3}(2n+1)}\right) \tag{190}$$

are the coordinates of the singularity. Substituting Equation 180 into Equation 179 and Equation 181 into Equation 178,

$$\theta = 2\pi \frac{a\left(m+\frac{1}{2}\right) + x' \pm \frac{a}{2\sqrt{3}}}{3a(2n+1)} \tag{191}$$

$$r = y' \pm \frac{a}{2\sqrt{3}} + a\left(n+\frac{1}{2}\right) . \tag{192}$$

Next, substitute Equations 182 and 183 into these two equations.

$$r = B_{21}x'' + B_{22}y'' \pm \frac{a}{2\sqrt{3}} + a\left(n+\frac{1}{2}\right) \tag{193}$$

$$\theta = 2\pi \frac{a\left(m+\frac{1}{2}\right) + B_{11}x'' + B_{12}y'' \pm \frac{a}{2\sqrt{3}}}{3a(2n+1)} \tag{194}$$

Finally, substitute the values from Equations 184 and 185 into these two equations.

$$r = B_{21}u + B_{22}uv \pm \frac{a}{2\sqrt{3}} + a\left(n+\frac{1}{2}\right) \tag{195}$$

$$\theta = 2\pi \frac{a\left(m+\frac{1}{2}\right) + B_{11}u + B_{12}uv \pm \frac{a}{2\sqrt{3}}}{3a(2n+1)} \tag{196}$$

For the first triangle, the inverse transformation matrix and Jacobian are

$$B_4^{(1)} = a\begin{pmatrix} -\frac{1}{6}(3+\sqrt{3}) & 0 \\ -\frac{1}{6}(3+\sqrt{3}) & 1 \end{pmatrix} \tag{197}$$

$$J_4^{(1)} = a^2 \frac{3+\sqrt{3}}{6} . \tag{198}$$

Next, substitute Equations 197 and 198 into Equations 195 and 196.

$$r = -\frac{3+\sqrt{3}}{6}u + uv \pm \frac{a}{2\sqrt{3}} + a\left(n+\frac{1}{2}\right) \tag{199}$$

$$\theta = 2\pi \frac{a\left(m+\frac{1}{2}\right) - \frac{3+\sqrt{3}}{6}u \pm \frac{a}{2\sqrt{3}}}{3a(2n+1)} \tag{200}$$

We can now compute the desired integral for this triangle.

$$\int rG(\rho)\,dr\,d\theta = \left[\frac{2\pi}{a(6n+3)}\right]\left(a^2\frac{3+\sqrt{3}}{6}\right)\int_0^1 u\,du \int_0^1 r(u,v)G(\rho(u,v))\,dv \tag{201}$$

## 5.9. Fortran Implementation

I have implemented the above algorithms in two modules, `patch.f95` and `alt_patch.f95`. I tested integrating over the patches using the pFUnit unit testing framework in the file `test_patch.pf`. These three files are included in their entirety in the appendix, but I'll discuss some details of each here.

### 5.9.1. patch.f95

The `patch` module is declared in the file `patch.f95`. It implements a `patch_par` class which has producedures for computing the radius of a point on the patch, $\theta$ of a point on a patch, the Greene's function, and the Jacobian. An object is created with the following syntax:

```
use patch
implicit none
integer, parameter :: n=3, m=2, triangle=1, singular_point=2
real(real64), parameter :: a=0.5d0
type(patch_par) pp

pp = patch_par(n, m, triangle, singular_point, a)
```

The arguments are as follows:

**n** The ring of the arc, the innermost ring being 0.

**m** The patch number within the arc. $0 \leq m \leq 6n + 2$

**triangle** The triangle within the arc, numbered as in this document.

**singular_point** The number of the singular point, numbered as in this document.

**a** The radial width of each patch.

The $B$ matrix needed for later computations in computed by explicit construction.

The `radius` procedure is called as follows:

```
pp = patch_par(n, m, triangle, singular_point, a)
pp%radius(xpp,ypp)
```

The parameters are the $x''$ and $y''$ coordinates from the integration. The routine returns the radial coordinates of the point on the patch with the origina being the center of the inner circle.

The `theta` procudure is called as follows:

```
pp = patch_par(n, m, triangle, singular_point, a)
pp%theta(xpp,ypp)
```

The arguments `xpp` and `ypp` have the same meaning as in the call to the `radius` procedure.

The `grfunc` procedure computes the Greene's function for this triangle and chosen singular point. It has the same arguments and calling procedure as `radius` and `theta`.

The `jacobian` takes no arguments. It has the Jacobian needed to integrate a function over the varibles $x''$ and $y''$ (or equivalently $u$ and $v$). It is called as follows:

```
pp = patch_par(n, m, triangle, singular_point, a)
pp%jacobian()
```

### 5.9.2. alt_patch.f95

The `alt_patch` module in the file `alt_patch.f95` has exactly the same calling convention as `patch`. The difference is that the $B$ matrix elements and jacobian are computed algorithmically using the alternative derivations in Section 5.5.

### 5.9.3. test_patch.pf

The `test_patch.pf` file has unit tests which test the internal computations in the `patch` and `alt_patch` modules by comparing them with independent Mathematica calculations.

The `Parameters` subroutine checks the initialization parameters in the `alt_patch` module apart from the $B$ matrix. The `altB` subroutine checks the $B$ matrix computed in the `alt_patch` module. The `ajtJ` subroutine checks the jacabian calculations in teh same module. The subroutines `patchPar` and `patchBJ` are the corresponding tests for the `patch` module. The subroutine `constInt` tests integration of a constant function over a patch and checks agreement with Mathematica. The subroutine GreeneInt does an integration of a Greene's function over a patch and compares it to Mathematica.

At this point, all of the unit tests pass successfully.

# 6. Application Notes

Our primary interest in a surface which is not flat is to integrate over a rough surface. Care should be taken in selecting the algorithm for generating the surface so that partial derivatives are easily and accurately evaluated. Two good choices are surfaces characterized by cubic splines and surfaces from band-width limited Fourier transforms.

## 6.1. Splines

Cubic splines are piece-wise cubic polynomials with continuous first and second partial derivatives. There are two simple ways I can think of to generate such surfaces. One would be to use a Gaussian random number generator to create random surface points which are more widely spaced than the patch separations. The splines will be forced to go through these points (called knots) exactly, but the other points will be interpolated. If the points are relatively far apart, the surface will be smooth with a spatial frequency equal to about 1/3 the spline separation (since the cubic could have three local extrema in general).

Alternatively, the spline could be generated by using a Gaussian random number generator at each grid point and then creating a smoothing spline to interpolate across the region. The smoothing spline has adjustable knots which are varied to minimize the deviation of the spline from the data points. It is could also controlled by a smoothing parameter which is varied to constrain discontinuities in the derivatives in adjoining regions.

The partial derivatives of the splines are easily computed since the interpolation is just a polynomial. Most spline libraries have facilities for computing the partial derivatives for you.

### 6.2. Fourier Transform

Our AFM data suggests that some surfaces are well-modeled by a noise function which has an envelop like a half-Gaussian in the Fourier domain. Such a surface can be generated by starting with points on the surface with random Gaussian noise added. This surface is then transformed to the frequency domain using a 2d Fast Fourier Transform. In that domain, the Fourier components are multiplied by

$$f(k) = e^{-k^2/2\sigma^2} \tag{202}$$

where $\sigma$ is a constant chosen to regulate the amount of smoothing and

$$k^2 = k_x^2 + k_y^2 \tag{203}$$

is the wave number (i.e. the independent variable in the Fourier domain. The filtered spectrum is then transformed back to the spatial domain using an inverse fast Fourier transform to generate the required heights.

The partial derivatives are easily computed since the Fourier expansion is just a sum of complex exponentials. One has to take care of high frequency ghosts in the Fourier transforms because of aliasing. I don't think this will be a problem in our case since we are applying a low-pass filter which will suppress them.

## A. Sample Julia Code

Here is a function applying this method in Julia.

```
"""
cint integrates the function f over a circle of radius r
using n rings. Each ring is divided into 3(2i+1) segments,
where i=0 for the inner most ring and can have values up to n-1.
"""
function cint(f,r,n)
    a=r/n
    sum=0.0
    for i=0:n-1
        ip=(2*i+1+1/sqrt(3.0))/(2*i+1)
        im=(2*i+1-1/sqrt(3.0))/(2*i+1)
        rp=a*(i+0.5+1/(2*sqrt(3.0)))
        rm=a*(i+0.5-1/(2*sqrt(3.0)))
        for m=0:6*i+2
            tp=2*pi*(m+0.5+1/(2*sqrt(3.0)))/(3*(2*i+1))
            tm=2*pi*(m+0.5-1/(2*sqrt(3.0)))/(3*(2*i+1))
            sum += im*f(rm,tm)+im*f(rm,tp)+ip*f(rp,tm)+ip*(f(rp,tp))
        end
    end
    sum *= pi*a^2/12
end
```

## B. patch.f95

```
! Routines and constants specialized to integration over a particular
! patch and triangle.
! Steve Turley, August 8, 2017
module patch
  use iso_fortran_env, only : real64
  implicit none
  private
  type patch_par
     ! private
     ! public
     integer :: n, m, triangle, singular_point
     real(real64) :: a
     real(real64) :: b(2,2), j
     real(real64) :: xs, ys
   contains
     procedure :: radius
     procedure :: theta
     procedure :: grfunc ! Greene's function
     procedure :: jacobian
  end type patch_par

  ! constructor(s)
  interface patch_par
     module procedure :: init
  end interface patch_par

  public :: patch_par

contains

  function init(n,m,triangle, singular_point, a)
    integer, intent(in) :: n, m, triangle, singular_point
    real(real64), intent(in) :: a
    type(patch_par) init
    real(real64) :: apt
    init%n = n
    init%m = m
    init%triangle = triangle
    init%a = a
    init%singular_point = singular_point
   ! Set singular points xs and ys
```

```
apt = a/(2*sqrt(3.0d0))
if(singular_point < 3) then
   init%xs = -apt
else
   init%xs = apt
end if
if(singular_point == 1 .OR. singular_point == 4) then
   init%ys = -apt
else
   init%ys = apt
end if
! This is where the B's and J's are initialized. It isn't final yet
select case (10*singular_point+triangle)
case(11)
   init%b = reshape((/ -a*(3-sqrt(3.d0))/6, -a*(3-sqrt(3.d0))/6,&
        0.0d0, a /), shape(init%b))
   init%j = a**2*(3-sqrt(3.d0))/6
case(21)
   init%b = reshape((/ -a*(3-sqrt(3.d0))/6, -a*(3+sqrt(3.d0))/6,&
        0.0d0, a /), shape(init%b))
   init%j = a**2*(3-sqrt(3.d0))/6
case(31)
   init%b = reshape((/ -a*(3+sqrt(3.d0))/6, -a*(3+sqrt(3.d0))/6,&
        0.0d0, a /), shape(init%b))
   init%j = a**2*(3+sqrt(3.d0))/6
case(41)
   init%b = reshape((/ -a*(3+sqrt(3.d0))/6, -a*(3-sqrt(3.d0))/6,&
        0.0d0, a /), shape(init%b))
   init%j = a**2*(3+sqrt(3.d0))/6
case(12)
   init%b = reshape((/ -a*(3-sqrt(3.d0))/6, a*(3+sqrt(3.d0))/6,&
        a, 0.d0 /), shape(init%b))
   init%j = a**2*(3+sqrt(3.d0))/6
case(22)
   init%b = reshape((/ -a*(3-sqrt(3.d0))/6, a*(3-sqrt(3.d0))/6,&
        a, 0.d0 /), shape(init%b))
   init%j = a**2*(3-sqrt(3.d0))/6
case(32)
   init%b = reshape((/ -a*(3+sqrt(3.d0))/6, a*(3-sqrt(3.d0))/6,&
        a, 0.d0 /), shape(init%b))
   init%j = a**2*(3-sqrt(3.d0))/6
case(42)
   init%b = reshape((/ -a*(3+sqrt(3.d0))/6, a*(3+sqrt(3.d0))/6,&
        a, 0.d0 /), shape(init%b))
```

```
      init%j = a**2*(3+sqrt(3.d0))/6
    case(13)
      init%b = reshape((/ a*(3+sqrt(3.d0))/6, a*(3+sqrt(3.d0))/6,&
           0.0d0, -a /), shape(init%b))
      init%j = a**2*(3+sqrt(3.d0))/6
    case(23)
      init%b = reshape((/ a*(3+sqrt(3.d0))/6, a*(3-sqrt(3.d0))/6,&
           0.0d0, -a /), shape(init%b))
      init%j = a**2*(3+sqrt(3.d0))/6
    case(33)
      init%b = reshape((/ a*(3-sqrt(3.d0))/6, a*(3-sqrt(3.d0))/6,&
           0.0d0, -a /), shape(init%b))
      init%j = a**2*(3-sqrt(3.d0))/6
    case(43)
      init%b = reshape((/ a*(3-sqrt(3.d0))/6, a*(3+sqrt(3.d0))/6,&
           0.0d0, -a /), shape(init%b))
      init%j = a**2*(3-sqrt(3.d0))/6
    case(14)
      init%b = reshape((/ a*(3+sqrt(3.d0))/6, -a*(3-sqrt(3.d0))/6,&
          -a, 0.d0 /), shape(init%b))
      init%j = a**2*(3-sqrt(3.d0))/6
    case(24)
      init%b = reshape((/ a*(3+sqrt(3.d0))/6, -a*(3+sqrt(3.d0))/6,&
          -a, 0.d0 /), shape(init%b))
      init%j = a**2*(3+sqrt(3.d0))/6
    case(34)
      init%b = reshape((/ a*(3-sqrt(3.d0))/6, -a*(3+sqrt(3.d0))/6,&
          -a, 0.d0 /), shape(init%b))
      init%j = a**2*(3+sqrt(3.d0))/6
    case(44)
      init%b = reshape((/ a*(3-sqrt(3.d0))/6, -a*(3-sqrt(3.d0))/6,&
          -a, 0.d0 /), shape(init%b))
      init%j = a**2*(3-sqrt(3.d0))/6
    end select
  end function init

  function radius(this, xpp, ypp)
    class(patch_par) this
    real(real64), intent(in) :: xpp, ypp
    real(real64)::radius
    radius = this%b(2,1)*xpp+this%b(2,2)*ypp+this%ys+this%a&
         *(this%n+0.5d0)
  end function radius
```

```
  function theta(this, xpp, ypp)
    class(patch_par) this
    real(real64), intent(in) :: xpp, ypp
    real(real64)::theta
    real(real64), parameter :: pi = 4.0d0*atan(1.0d0)
    theta = 2*pi*(this%a*(this%m+0.5d0)+this%b(1,1)*xpp+&
         this%b(1,2)*ypp+this%xs)/(3*this%a*(2*this%n+1))
  end function theta

  function grfunc(this, xpp, ypp)
    class(patch_par) this
    real(real64), intent(in) :: xpp, ypp
    complex(real64)::grfunc
    real(real64), parameter :: pi = 4.0d0*atan(1.0d0)
    real(real64), parameter :: k = 2*pi
    real(real64) :: rho, r, th, rs, ths
    r = this%radius(xpp, ypp)
    th = this%theta(xpp, ypp)
    rs = this%ys + this%a*(this%n+0.5d0)
    ths = 2*pi*(this%a*(this%m+0.5d0)+this%xs)/(3*this%a*(2*this%n+1))
    rho = sqrt(r**2+rs**2-2*r*rs*cos(th-ths))
    grfunc = exp(cmplx(0.d0,k*rho,real64)) /(4*pi*rho)
  end function grfunc

  function jacobian(this)
    class(patch_par) this
    real(real64)::jacobian
    real(real64), parameter :: pi = 4.0d0*atan(1.0d0)
    jacobian = this%j*2*pi/(this%a*(6*this%n+3))
  end function jacobian

end module patch
```

## C. alt_patch.f95

```
! Routines and constants specialized to integration over a particular
! patch and triangle.
! Steve Turley, August 10, 2017
!
! This is an alternative to the patch module which changes the numbering
! of the singular points and uses an algorithmic approach to setting
! the b and j variables. The b's and j's are different than those computed
! by the patch module.
!
```

```
module alt_patch
  use iso_fortran_env, only : real64
  implicit none
  private
  type patch_par
     ! private
     ! public
     integer :: n, m, triangle, singular_point
     real(real64) :: a
     real(real64) :: b(2,2), j
     real(real64) :: xs, ys
   contains
     procedure :: radius
     procedure :: theta
     procedure :: grfunc ! Greene's function
     procedure :: jacobian
  end type patch_par

  ! constructor(s)
  interface patch_par
     module procedure :: init
  end interface patch_par

  public :: patch_par

contains

  function init(n,m,triangle, singular_point, a)
    integer, intent(in) :: n, m, triangle, singular_point
    real(real64), intent(in) :: a
    type(patch_par) init
    real(real64) :: apt, pp2, qq2, pp3, qq3
    init%n = n
    init%m = m
    init%triangle = triangle
    init%a = a
    init%singular_point = singular_point
    ! Set singular points xs and ys
    apt = a/(2*sqrt(3.0d0))
    if(singular_point < 3) then
       init%xs = -apt
    else
       init%xs = apt
    end if
```

```
  if(singular_point == 1 .OR. singular_point == 4) then
     init%ys = -apt
  else
     init%ys = apt
  end if
 ! This is where the B's and J's are initialized.
  pp2 = p2(a, triangle,singular_point)
  pp3 = p3(a, triangle,singular_point)
  qq2 = q2(a, triangle,singular_point)
  qq3 = q3(a, triangle,singular_point)
  init%b=reshape([pp2,qq2,pp3-pp2,qq3-qq2],shape(init%b))
  init%j=adet(init%b)
end function init

function p2(a, t,s)
  real(real64), intent(in) :: a
  integer, intent(in) :: t,s
  real(real64) :: p2
  integer :: s1, s2
  if(t<3) then
     s1=-1
  else
     s1=1
  end if
  if(s<3) then
     s2=-1
  else
     s2=1
  end if
  p2=a*(s1/2.0d0-s2/(2*sqrt(3.d0)))
end function p2

function q2(a, t, s)
  real(real64), intent(in) :: a
  integer, intent(in) :: t,s
  real(real64) :: q2
  integer :: s1, s2
  if(t==1 .OR. t==4) then
     s1=-1
  else
     s1=1
  end if
  if(s==1 .OR. s==4) then
     s2=-1
```

```
  else
     s2=1
  end if
  q2=a*(s1/2.0d0-s2/(2*sqrt(3.d0)))
end function q2

function p3(a, t, s)
  real(real64), intent(in) :: a
  integer, intent(in) :: t,s
  real(real64) :: p3
  integer :: s1, s2
  if(t==1 .OR. t==4) then
     s1=-1
  else
     s1=1
  end if
  if(s<3) then
     s2=-1
  else
     s2=1
  end if
  p3=a*(s1/2.0d0-s2/(2*sqrt(3.d0)))
end function p3

function q3(a, t,s)
  real(real64), intent(in) :: a
  integer, intent(in) :: t,s
  real(real64) :: q3
  integer :: s1, s2
  if(t<3) then
     s1=1
  else
     s1=-1
  end if
  if(s==1 .OR. s==4) then
     s2=-1
  else
     s2=1
  end if
  q3=a*(s1/2.0d0-s2/(2*sqrt(3.d0)))
end function q3

function adet(b)
  real(real64), intent(in) :: b(2,2)
```

```
    real(real64) :: adet
    adet = abs(b(1,1)*b(2,2)-b(1,2)*b(2,1))
  end function adet

  function radius(this, xpp, ypp)
    class(patch_par) this
    real(real64), intent(in) :: xpp, ypp
    real(real64)::radius
    radius = this%b(2,1)*xpp+this%b(2,2)*ypp+this%ys+this%a&
         *(this%n+0.5d0)
  end function radius

  function theta(this, xpp, ypp)
    class(patch_par) this
    real(real64), intent(in) :: xpp, ypp
    real(real64)::theta
    real(real64), parameter :: pi = 4.0d0*atan(1.0d0)
    theta = 2*pi*(this%a*(this%n+0.5d0)+this%b(1,1)*xpp+&
         this%b(1,2)*ypp+this%xs)/(3*this%a*(2*this%n+1))
  end function theta

  function grfunc(this, xpp, ypp)
    class(patch_par) this
    real(real64), intent(in) :: xpp, ypp
    complex(real64)::grfunc
    real(real64), parameter :: pi = 4.0d0*atan(1.0d0)
    real(real64), parameter :: k = 2*pi
    real(real64) :: rho, r, th, rs, ths
    r = this%radius(xpp, ypp)
    th = this%theta(xpp, ypp)
    rs = this%ys + this%a*(this%n+0.5d0)
    ths = 2*pi*(this%a*(this%n+0.5d0)+this%xs)/(3*this%a*(2*this%n+1))
    rho = sqrt(r**2+rs**2-2*r*rs*cos(th-ths))
    grfunc = exp(cmplx(0.d0,k*rho,real64)) /(4*pi*rho)
  end function grfunc

  function jacobian(this)
    class(patch_par) this
    real(real64)::jacobian
    real(real64), parameter :: pi = 4.0d0*atan(1.0d0)
    jacobian = this%j*2*pi/(this%a*(6*this%n+3))
  end function jacobian

end module alt_patch
```

## D. test_patch.pf

```
@test
subroutine Parameters
  ! Test the patch parameters
  use iso_fortran_env
  ! use patch
  use alt_patch
  use pfunit_mod
  implicit none
  integer, parameter :: n=3, m=2, triangle=1, singular_point=2
  real(real64), parameter :: a=0.5d0, xpp=0.5d0, ypp=0.5d0
  type(patch_par) pp
  real(real64) :: xs, ys, apt
  integer :: sp
  pp = patch_par(n, m, triangle, singular_point, a)
  @assertEqual(n, pp%n)
  @assertEqual(m, pp%m)
  @assertEqual(a, pp%a,1d-15)
  apt = a/(2*sqrt(3.0d0))
  do sp=1,4
     pp = patch_par(n,m,triangle, sp, a)
     select case(sp)
     case(1)
        xs=-apt
        ys=-apt
     case(2)
        xs=-apt
        ys=apt
     case(3)
        xs=apt
        ys=apt
     case(4)
        xs=apt
        ys=-apt
     end select
     @assertEqual(xs,pp%xs, 1d-14)
     @assertEqual(ys,pp%ys, 1d-14)
  end do
end subroutine Parameters

@test
subroutine altB
```

```
! Test the patch parameters
use iso_fortran_env
use alt_patch
use pfunit_mod
implicit none
integer, parameter :: n=3, m=2
real(real64), parameter :: a=0.5d0
type(patch_par) pp
real(real64), parameter :: bm = a*(3-sqrt(3.0d0))/6
real(real64), parameter :: bp = a*(3+sqrt(3.0d0))/6
! Check B
pp = patch_par(n, m, 1, 1, a)
@assertEqual(-bm, pp%b(1,1),1d-14)
@assertEqual(-bm, pp%b(2,1),1d-14)
@assertEqual(a,pp%b(2,2),1d-14)
@assertEqual(0.d0,pp%b(1,2),1d-14)
pp = patch_par(n,m,1,2,a)
@assertEqual(-bm, pp%b(1,1),1d-14)
@assertEqual(0.d0, pp%b(1,2), 1d-14)
@assertEqual(-bp, pp%b(2,1), 1d-14)
@assertEqual(a, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 1, 3, a)
@assertEqual(-bp, pp%b(1,1), 1d-14)
@assertEqual(0.d0, pp%b(1,2), 1d-14)
@assertEqual(-bp, pp%b(2,1), 1d-14)
@assertEqual(a, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 1, 4, a)
@assertEqual(-bp, pp%b(1,1), 1d-14)
@assertEqual(0.d0, pp%b(1,2), 1d-14)
@assertEqual(-bm,pp%b(2,1), 1d-14)
@assertEqual(a, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 2, 1, a)
@assertEqual(-bm, pp%b(1,1), 1d-14)
@assertEqual(a, pp%b(1,2),1d-14)
@assertEqual(bp, pp%b(2,1), 1d-14)
@assertEqual(0.d0,pp%b(2,2), 1d-14)
pp = patch_par(n, m, 2, 2, a)
@assertEqual(-bm, pp%b(1,1), 1d-14)
@assertEqual(a, pp%b(1,2), 1d-14)
@assertEqual(bm, pp%b(2,1), 1d-14)
@assertEqual(0.d0, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 2, 3, a)
@assertEqual(-bp, pp%b(1,1), 1d-14)
@assertEqual(a, pp%b(1,2), 1d-14)
```

```
@assertEqual(bm, pp%b(2,1), 1d-14)
@assertEqual(0.d0, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 2, 4, a)
@assertEqual(-bp, pp%b(1,1), 1d-14)
@assertEqual(a, pp%b(1,2), 1d-14)
@assertEqual(bp, pp%b(2,1), 1d-14)
@assertEqual(0.d0, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 3, 1, a)
@assertEqual(bp, pp%b(1,1), 1d-14)
@assertEqual(0.d0, pp%b(1,2), 1d-14)
@assertEqual(bp, pp%b(2,1), 1d-14)
@assertEqual(-a, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 3, 2, a)
@assertEqual(bp, pp%b(1,1), 1d-14)
@assertEqual(0.d0, pp%b(1,2), 1d-14)
@assertEqual(bm, pp%b(2,1), 1d-14)
@assertEqual(-a, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 3, 3, a)
@assertEqual(bm, pp%b(1,1), 1d-14)
@assertEqual(0.d0, pp%b(1,2), 1d-14)
@assertEqual(bm,pp%b(2,1), 1d-14)
@assertEqual(-a, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 3, 4, a)
@assertEqual(bm, pp%b(1,1), 1d-14)
@assertEqual(0.d0, pp%b(1,2), 1d-14)
@assertEqual(bp, pp%b(2,1), 1d-14)
@assertEqual(-a, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 4, 1, a)
@assertEqual(bp, pp%b(1,1), 1d-14)
@assertEqual(-a, pp%b(1,2), 1d-14)
@assertEqual(-bm, pp%b(2,1), 1d-14)
@assertEqual(0.d0, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 4, 2, a)
@assertEqual(bp, pp%b(1,1), 1d-14)
@assertEqual(-a, pp%b(1,2), 1d-14)
@assertEqual(-bp, pp%b(2,1), 1d-14)
@assertEqual(0.d0, pp%b(2,2), 1d-14)
pp = patch_par(n, m, 4, 3, a)
@assertEqual(bm, pp%b(1,1), 1d-14)
@assertEqual(-a, pp%b(1,2), 1d-14)
@assertEqual(-bp, pp%b(2,1), 1d-14)
@assertEqual(0.d0, pp%b(2,2), 1d-14)
pp = patch_par(n,m,4,4,a)
@assertEqual(bm, pp%b(1,1), 1d-14)
```

```
  @assertEqual(-a, pp%b(1,2), 1d-14)
  @assertEqual(-bm, pp%b(2,1), 1d-14)
  @assertEqual(0.d0, pp%b(2,2), 1d-14)
end subroutine altB

@test
subroutine altJ
  ! Test the patch parameters
  use iso_fortran_env
  use alt_patch
  use pfunit_mod
  implicit none
  integer, parameter :: n=3, m=2
  real(real64), parameter :: a=0.5d0
  type(patch_par) pp
  integer :: sp, tr
  real(real64), parameter :: bm = a*(3-sqrt(3.0d0))/6
  real(real64), parameter :: bp = a*(3+sqrt(3.0d0))/6
  real(real64), parameter :: jm = bm*a
  real(real64), parameter :: jp = bp*a
  character(80) :: msg
  ! Check J
  pp = patch_par(n, m, 1, 1, a)
  @assertEqual(jm, pp%j,1d-14)
  pp = patch_par(n,m,1,2,a)
  @assertEqual(jm, pp%j, 1d-14)
  pp = patch_par(n, m, 1, 3, a)
  @assertEqual(jp, pp%j, 1d-14)
  pp = patch_par(n, m, 1, 4, a)
  @assertEqual(jp, pp%j, 1d-14)
  pp = patch_par(n, m, 2, 1, a)
  @assertEqual(jp, pp%j, 1d-14)
  pp = patch_par(n, m, 2, 2, a)
  @assertEqual(jm, pp%j, 1d-14)
  pp = patch_par(n, m, 2, 3, a)
  @assertEqual(jm, pp%j, 1d-14)
  pp = patch_par(n, m, 2, 4, a)
  @assertEqual(jp, pp%j, 1d-14)
  pp = patch_par(n, m, 3, 1, a)
  @assertEqual(jp, pp%j, 1d-14)
  pp = patch_par(n, m, 3, 2, a)
  @assertEqual(jp, pp%j, 1d-14)
  pp = patch_par(n, m, 3, 3, a)
  @assertEqual(jm, pp%j, 1d-14)
```

```
  pp = patch_par(n, m, 3, 4, a)
  @assertEqual(jm, pp%j, 1d-14)
  pp = patch_par(n, m, 4, 1, a)
  @assertEqual(jm, pp%j, 1d-14)
  pp = patch_par(n, m, 4, 2, a)
  @assertEqual(jp, pp%j, 1d-14)
  pp = patch_par(n, m, 4, 3, a)
  @assertEqual(jp, pp%j, 1d-14)
  pp = patch_par(n,m,4,4,a)
  @assertEqual(jm, pp%j, 1d-14)
  do tr=1,4
     do sp=1,4
        pp = patch_par(n,m,tr,sp,a)
        write(msg, fmt='("for triangle ",i1," and s.p. ",i1)')tr,sp
        @assertEqual(adet(pp%b), pp%j,1d-14,msg)
     end do
  end do
contains
  function adet(b)
    use iso_fortran_env, only: real64
    implicit none
    real(real64), intent(in) :: b(2,2)
    real(real64) :: adet
    adet=abs(b(1,1)*b(2,2)-b(1,2)*b(2,1))
  end function adet
end subroutine altJ

@test
subroutine patchPar
  ! Test the patch parameters
  use iso_fortran_env
  use alt_patch, only : apatch=>patch_par
  use patch, only: ppatch=>patch_par
  use pfunit_mod
  implicit none
  real(real64), parameter :: a=0.5d0
  type(ppatch) pp
  type(apatch) ap
  integer, parameter :: n=3, m=2, triangle=1, singular_point=2
  integer :: sp, tr

  pp = ppatch(n, m, triangle, singular_point, a)
  ap = apatch(n, m, triangle, singular_point, a)
  @assertEqual(ap%n, pp%n)
```

```
  @assertEqual(ap%m, pp%m)
  @assertEqual(ap%a, pp%a,1d-15)
  do tr=1,4
     do sp=1,4
        pp = ppatch(n, m, tr, sp, a)
        ap = apatch(n, m, tr, sp, a)
        @assertEqual(ap%xs,pp%xs, 1d-14)
        @assertEqual(ap%ys,pp%ys, 1d-14)
     end do
  end do
end subroutine patchPar

@test
subroutine patchBJ
  ! Test the patch parameters
  use iso_fortran_env
  use alt_patch, only : apatch=>patch_par
  use patch, only: ppatch=>patch_par
  use pfunit_mod
  implicit none
  real(real64), parameter :: a=0.5d0
  type(ppatch) pp
  type(apatch) ap
  integer, parameter :: n=3, m=2
  integer :: sp, tr, bx, by
  character(80) :: msg

  do tr=1,4
     do sp=1,4
        pp = ppatch(n,m,tr,sp,a)
        ap = apatch(n,m,tr,sp,a)
        write(msg, fmt='("J for triangle ",i1," and s.p. ",i1)')tr,sp
        @assertEqual(ap%j, pp%j, 1d-14, trim(msg))
        do bx=1,2
           do by=1,2
              write(msg, fmt='("b for tr",i1," sp",i1," bx",i1," by",i1)')tr,s
              @assertEqual(ap%b(bx,by),pp%b(bx,by), 1d-14, trim(msg))
           end do
        end do
     end do
  end do
end subroutine patchBJ

@test
```

```
subroutine constInt
  use iso_fortran_env, only : real64
  use alt_patch
  use duffy
  use pfunit_mod
  implicit none
  integer, parameter :: n=3, m=2, triangle=1, singular_point=2
  real(real64), parameter :: a=0.50, xpp=0.5d0, ypp=0.5d0
  real(real64) :: rad, the, jac, rint
  complex(real64) :: grf
  real(real64), parameter :: mradius = 1.94716878364870d0 ! from Mathematica
  real(real64), parameter :: mtheta=0.630012726620808d0 ! from Mathematica
  complex(real64), parameter :: mgreene=cmplx(0.865006682017783d0,0.4789609952
  real(real64), parameter :: mjacobian=0.0316141259370375d0 ! Mathematica
  real(real64), parameter :: apt = a/(2*sqrt(3.d0))
  real(real64), parameter :: mconst = 0.261799d0
  type(patch_par) pp
  integer :: t

  pp = patch_par(n, m, triangle, singular_point, a)
  rad = pp%radius(xpp, ypp)
  @assertEqual(mradius,rad,1d-14,"radius")
  the = pp%theta(xpp, ypp)
  @assertEqual(mtheta,the,1d-14,"theta")
  @assertEqual(-apt,pp%xs,1d-14,"xs")
  @assertEqual(apt,pp%ys,1d-14,"ys")
  grf = pp%grfunc(xpp, ypp)
  ! interesting that the next test requires this loose of a tolerance
  @assertEqual(mgreene,grf,1d-13)
  jac = pp%jacobian()
  @assertEqual(mjacobian,jac,1d-14,"jacobian")

  ! Diagnostics using simple constant integral
  rint = 0.d0
  do t=1,4
     pp = patch_par(n, m, t, singular_point, a)
     rint = rint+duffy_int(const, 1d-10)*pp%jacobian()
  end do
  @assertEqual(mconst,rint,1d-6,"constant integral")

contains

  function const(x,y)
    real(real64), intent(in) :: x,y
```

```
    real(real64) :: const
    const = pp%radius(x,y)
  end function const

end subroutine constInt

@test
subroutine GreeneInt
  use iso_fortran_env, only : real64
  use alt_patch
  use duffy
  use pfunit_mod
  implicit none
  integer, parameter :: n=3, m=2, singular_point = 2
  real(real64), parameter :: a=0.50, xpp=0.5d0, ypp=0.5d0
  complex(real64) :: gint
  complex(real64), parameter :: mint=cmplx(0.0489868, 0.0731842, real64)
  real(real64), parameter :: apt = a/(2*sqrt(3.d0))
  type(patch_par) pp
  integer :: t

  gint = 0.d0
  do t=1,4
     pp = patch_par(n, m, t, singular_point, a)
     gint = gint + duffy_int(cdf, 1d-10)*pp%jacobian()
  end do
  @assertEqual(mint, gint, 1d-6, "constant with Greene function integral")
contains

  function cdf(x,y)
    real(real64), intent(in) :: x,y
    complex(real64) :: cdf
    cdf = pp%radius(x,y)*pp%grfunc(x,y)
  end function cdf

end subroutine GreeneInt
```

## References


[1] National Institute of Standards and Technology, Digital Library of Mathematical Functions (http://dlmf.nist.gov/3.5#x, accessed June 21, 2017).