SOI-MDI Technical Notes

Generation of Artificial Test Data I. Individual Images

R. Bogart

October 1992

SOI-TN-095

Stanford - Lockheed Institute
for Astrophysical and Space Research

Generation of Artificial Test Data I. Individual Images

R. S. Bogart

SOI Technical Note TN-92-095 *[incomplete draft]*

This note describes the library (and pipeline) routine SolarView, a program for the generation of pseudo-Dopplergrams and other data in the format of solar image data as observed by MDI or other comparable instruments. The essential elements of SolarView are accurately projected values of surface spherical harmonics that may be combined with known or assumed contributions to the observed signal, such as rotation, large- and small-scale motions, limb shifts, limb darkening, instrumental gradients and distortions, and random noise.

1.0 Introduction

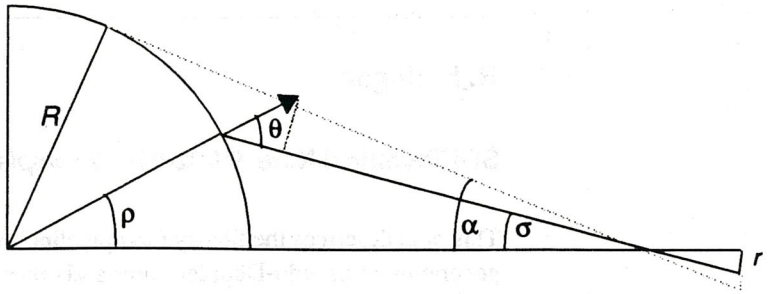
In order to test and evaluate procedures for the analysis of helioseismic data as part of the Solar Oscillations Investigation it is useful to have artifical data sets of known characteristics that match the expected character of the observed data as closely as possible. The function SolarView is designed to produce the equivalent of individual solar images as they would be observed in either the Doppler shift or intensity modes of the instrument. Physical phenomena included (or planned) include:

- Surface waves, such as *p*-mode amplitudes (as well as *f*- and *g*-modes).
- Surface flows, such as differential rotation, meridional flows, jets, and giant cells.
- Incoherent surface motions, including granulation and super-granulation.
- Parametrized effects, including limb-shift and limb darkening.
- Systematic effects, including instrumental gradients and distortion.
- Shot noise

2.0 Projection and Mapping

SolarView works by taking each pixel location or sub-pixel location for quadrature in turn and projecting that location backward onto an observed solar disc of epoch and likewise onto reference heliographic coordinates.

FIGURE 1.



2.1 Scalar Projection (Mapping)

Referring to Figure 1, the accurate mapping of a radius vector ***r*** on the plate into a projected radial angle ρ on a spherical sun is given by

$$\rho = \text{asin}\left(\frac{\sin\sigma}{\sin\theta}\right) - \sigma \qquad \text{(EQ 1)}$$

where σ is the angular projection of ***r***, given by

$$\sigma = \text{atan}\left(\frac{r}{r_S}\tan\alpha\right), \qquad \text{(EQ 2)}$$

r/r_S is the radial distance on the plate (in units of the projected image semidiameter) from the center of the projected image, which of course does not necessarily coincide with the plate center, and α is the angular semidiameter of the sun which depends on the distance of the observer from the sun centre. The azimuthal angle of the projected point from some arbitrary axis, typically the Carrington rotation axis, is simply given by the arithmetic sum of the azimuthal angle of the pixel location on the plate and the angle between the plate coordinate system and the projection of the arbitrary solar axis perpendicular to the sun centre–observer line.

For observations made at a distance of about $1\,\text{AU} \sim 2 \times 10^2 R$, α is of order 5×10^{-3}, so the plate curvature in Eqn. 2 is usually be neglected and small angle formulæ used:

$$\sigma \approx \frac{r}{r_S}\alpha \qquad \text{(EQ 3)}$$

$$\rho \approx \text{asin}\frac{r}{r_S} - \sigma \qquad \text{(EQ 4)}$$

SolarView, however, does not make these approximations.

2.2 Vector Projection

2.2.1 Line-of-sight components of a radial vector

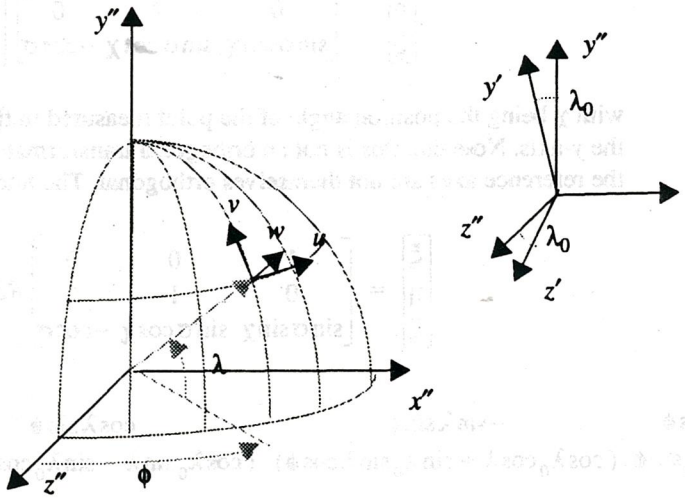
Let σ be the angle between the radial vector at the projected surface point and the line of sight. The projection of the vector $S\hat{r}$ in the line of sight is given by

$$S\cos\theta = S\cos(\rho+\sigma) = S\cos\left(\operatorname{asin}\left(\frac{\sin\sigma}{\sin\alpha}\right)\right) \qquad \text{(EQ 5)}$$

2.2.2 General components of a vector

Consider first the right-hand rectangular coordinate system (x'', y'', z'') oriented so that y'' coincides with the heliographic axis and z'' is the intersection of the plane containing the observer and the axis with the heliographic equator. The spherical coordinate unit vectors $(\hat{u}, \hat{v}, \hat{w})$ at a point on the surface of radius $r = R$, latitude λ, and longitude ϕ (measured in the direction of positive x', westward on the sky) are defined to coincide with the local direction vectors $(\hat{\phi}, \hat{\lambda}, \hat{r})$ (see Figure 2).

FIGURE 2.



The components of the vector $u\hat{\phi} + v\hat{\lambda} + w\hat{r}$ in the rectangular coordinate system $\xi''\hat{x}'' + \eta''\hat{y}'' + \zeta''\hat{z}''$ are given by:

$$\begin{bmatrix}\xi''\\ \eta''\\ \zeta''\end{bmatrix} = \begin{bmatrix}\cos\phi & -\sin\lambda\sin\phi & \cos\lambda\sin\phi\\ 0 & \cos\lambda & \sin\lambda\\ -\sin\phi & -\sin\lambda\cos\phi & \cos\lambda\cos\phi\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \equiv S\begin{bmatrix}u\\ v\\ w\end{bmatrix} \qquad \text{(EQ 6)}$$

Now consider the coordinate system (x', y', z') in which $\hat{x}'$ coincides with $\hat{x}''$ and $\hat{z}'$ points from the sun centre to the observer, *i.e.* a system rotated by the latitude of observed disc centre λ_0. Clearly,

$$\begin{bmatrix}\xi'\\ \eta'\\ \zeta'\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\lambda_0 & -\sin\lambda_0\\ 0 & \sin\lambda_0 & \cos\lambda_0\end{bmatrix}\begin{bmatrix}\xi''\\ \eta''\\ \zeta''\end{bmatrix} \equiv R\begin{bmatrix}\xi''\\ \eta''\\ \zeta''\end{bmatrix}$$

$$= \begin{bmatrix}\cos\phi & -\sin\lambda\sin\phi & \cos\lambda\sin\phi\\ \sin\lambda_0\sin\phi & (\cos\lambda_0\cos\lambda + \sin\lambda_0\sin\lambda\cos\phi) & (\cos\lambda_0\sin\lambda - \sin\lambda_0\cos\lambda\cos\phi)\\ -\cos\lambda_0\sin\phi & (\sin\lambda_0\cos\lambda - \cos\lambda_0\sin\lambda\cos\phi) & (\sin\lambda_0\sin\lambda + \cos\lambda_0\cos\lambda\cos\phi)\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \quad \textbf{(EQ 7)}$$

This is almost the transformation that is required, but not quite. ζ' is the (opposite of the) line-of-sight velocity for the center of the disc. To determine the line-of-sight velocity for points off the center, it is necessary to project onto the viewing angle by $\cos\sigma$. Let us define $\vec{v} \equiv \xi\hat{x} + \eta\hat{y} + \zeta\hat{z}$ with $\hat{x} = \hat{x}'$, $\hat{y} = \hat{y}'$, and $\hat{z}$ defined as the unit vector pointing *from* the observer to the point (R, λ, ϕ) (in order to support the convention that velocities are positive away from the observer), *i.e.*

$$\begin{bmatrix}\xi\\ \eta\\ \zeta\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ \sin\sigma\sin\chi & \sin\sigma\cos\chi & -\cos\sigma\end{bmatrix}\begin{bmatrix}\xi'\\ \eta'\\ \zeta'\end{bmatrix} \quad \textbf{(EQ 8)}$$

with χ being the position angle of the point measured in the positive x direction from the y-axis. Note that this is not an orthogonal transformation, except when $\sigma = 0$, as the reference axes are not themselves orthogonal. The final transformation is thus

$$\begin{bmatrix}\xi\\ \eta\\ \zeta\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ \sin\sigma\sin\chi & \sin\sigma\cos\chi & -\cos\sigma\end{bmatrix} RS\begin{bmatrix}u\\ v\\ w\end{bmatrix} \quad \textbf{(EQ 9)}$$

$$\begin{bmatrix}\xi\\ \eta\end{bmatrix} = \begin{bmatrix}\cos\phi & -\sin\lambda\sin\phi & \cos\lambda\sin\phi\\ \sin\lambda_0\sin\phi & (\cos\lambda_0\cos\lambda + \sin\lambda_0\sin\lambda\cos\phi) & (\cos\lambda_0\sin\lambda - \sin\lambda_0\cos\lambda\cos\phi)\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \quad \textbf{(EQ 10)}$$

$$\zeta = (\sin\sigma\sin\chi\cos\phi + \sin\sigma\cos\chi\sin\lambda_0\sin\phi + \cos\sigma\cos\lambda_0\sin\phi)\,u +$$
$$\{\sin\sigma\,[-\sin\chi\sin\lambda\sin\phi + \cos\chi\,(\cos\lambda_0\cos\lambda + \sin\lambda_0\sin\lambda\cos\phi)\,] - \cos\sigma\,(\sin\lambda_0\cos\lambda - \cos\lambda_0\sin\lambda\cos\phi)\}\,v +$$
$$\{\sin\sigma\,[\sin\chi\cos\lambda\sin\phi + \cos\chi\,(\cos\lambda_0\sin\lambda - \sin\lambda_0\cos\lambda\cos\phi)\,] - \cos\sigma\,(\sin\lambda_0\sin\lambda + \cos\lambda_0\cos\lambda\cos\phi)\}\,w \quad \textbf{(EQ 11)}$$

This transformation can be substantially simplified by using the spherical trigonometric identities:

$$\sin\lambda = \sin\lambda_0\cos\rho + \cos\lambda_0\sin\rho\cos\chi \quad \textbf{(EQ 12)}$$

$$\cos\lambda\sin\phi = \sin\rho\sin\chi \quad \textbf{(EQ 13)}$$

$$\cos\lambda\cos\phi = \cos\rho\cos\lambda_0 - \sin\lambda_0\sin\rho\cos\chi \quad \textbf{(EQ 14)}$$

so that we can write

$$\begin{bmatrix} \xi \\ \eta \\ \zeta \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\lambda\sin\phi & \sin\rho\sin\chi \\ \sin\lambda_0\sin\phi & (\cos\lambda_0\cos\lambda + \sin\lambda_0\sin\lambda\cos\phi) & (\cos\lambda_0\sin\lambda - \sin\lambda_0\cos\lambda\cos\phi) \\ \dfrac{\sin\theta\sin\chi\cos\lambda_0}{\cos\lambda} & \dfrac{\sin\theta\,(\cos\chi\cos\rho\cos\lambda_0 - \sin\rho\sin\lambda_0)}{\cos\lambda} & -\cos\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad \text{(EQ 15)}$$

which agrees, except for the conventional sign reversal, with Eq. 5 in the more restricted case $u = v = 0$.

2.3 Quadrature

In order to generate artificial data that effectively mimics data observed by integration of the light arriving within a detector aperture (a finite pixel) over an infinitesimal time interval, , it is necessary to integrate the signal over the aperture. In general this can only be done by quadrature of the values sampled at selected points, as the functional form of the data is very complex or unknown. Note that there is still no contribution of diffraction; we assume infinite resolution. It is assumed that the individual pixels are square, although they may be of arbitrary size (arbitrary magnification) and laid out in any rectangle pointed in any direction at an arbitrary position angle with respect to solar north. It is possible in the program to have a non-unity filling factor, *i.e.*an interpixel gap which receives no light. In practice, the interpixel gap on a CCD is zero: every photon falls on some pixel.

The physical observables of interest are determined by measuring the relative intensities of light in various (small) wavelength intervals. It is thus appropriate to weight them by a limb darkening function. In principle and practice the limb darkening function varies with wavelength. Also, the observables of interest may be formed by the linear combination of light from several wavelength intervals obeying different limb-darkening functions, and the appropriate weighting function for the observable may consequently differ from that for intensity. These effects are presently ignored. We assume that the pixels are sufficiently small that the appropriate weighting functions are nearly linear over any single pixel, so that a single representative weighting function, the limb darkening curve, may be used.

SolarView implements a nine-point fourth-order quadrature scheme unless the convergence criterion, selected as an input parameter, is zero or negative. The quadrature samples at the center with a numerical weight of 16, at each of the mid-points of the edges with weight of 4, and at each of the corners with weight of 1. (If the convergence criterion is non-positive, then only the center of the pixel is sampled.)

As long as we confine the modeled physical phenomena to well-defined, smoothly varying, and non-singular functions on the surface of a sphere we can expect convergence to a fixed value as the size of the pixel is arbitrarily decreased, at least as far as numerical accuracy permits. This suggests that the accuracy can be improved by subdividing each region of interest into sub-pixels over which a similar quadrature is performed, and continuing to do so until the resulting value has converged within some acceptable limit. For this purpose it is convenient to intially calculate the function at two levels of accuracy to decide whether iterative quadrature required. Nine-point quadratures commend themselves as providing the zeroth-order estimate (sampling at the center point) at no extra computational cost, provided that the center of the pixel is within the observed disc. Adding a fifth center point to a four-point quadrature method is still somewhat more efficient, however, although use of the legendre_by_x

() routine for the most computationally intensive part of the calculation means that the total computational time scales as less than the number of sampling points. In principle, five of the other nine points could be retained from calculations of previous rows and columns. For a large array, in which most pixels are not at edges, this means that only four new values would typically need to be calculated for each pixel, as opposed to five for a four-point quadrature and single point convergence comparator. (In practice such an algorithm has not yet been implemented; it assumes that the inter-pixel gap is zero, and that is still a changeable parameter in the program.)

In order to investigate the best schemes for quadrature and convergence I have considered a one-dimensional cut across the diameter at an angle of 30 degrees with respect to the equator in a model of observed line-of-sight component of velocities incorporating standard limb darkening, rotation, limb-shift, meridional circulation, supergranulation, and 300 m/sec radial components of each of seven spherical harmonic modes ranging from l=6 to l=768 (Fig 6.) In the absence of the modes, more than 99% of the pixels on the disc are converged to zeroth order. Because of this, no actual convergence scheme is yet implemented; convergence is merely checked.

3.0 Modeling of Physical Effects

3.1 Limb Darkening

Because the emission of radiation in the solar atmosphere is non-isotropic, the curvature of the surface induces geometric variations in the intensity of light (limb darkening). As remarked above (2.3), the program uses a single limb-darkening function as the weighting function for all physical observables. The function is of course also used as a multiplier when the selected observable is intensity. Following Pierce & Slaughter (1977) the limb-darkening function is parametrized as a polynomial in $\xi = \ln \mu$, where $\mu = \cos \theta$ is the cosine of the angle between the direction to the observer and the local normal for the point in question. Coefficients from first up through fifth-order in the ξ expansion can be supplied, but it is assumed that the zero-order coefficient is unity. Representative values from Pierce & Slaughter for a wavelength of 6791.40 Å are as follows:

$$\alpha_1 = 0.41708,\ \alpha_2 = 0.12512,\ \alpha_3 = 0.02825,\ \alpha_4 = 0.00528,\ \alpha_5 = 0.00051$$

The coefficients for a best-fit second-order polynomial at the same wavelength are

$$\alpha_1 = 0.36693,\ \alpha_2 = 0.05949$$

(This wavelength, which is the closest tabulated wavelength to the 6768Å Ni line selected for observations by GONG and SOI happens to also be the line which Pierce & Slaughter highlighted as having particularly favorable behaviour near the limb, so that its deconvolution was reliable.) The limb-darkening corresponding to the fifth-order polynomial, which is adopted as a standard, is shown in Figure 1.

3.2 Differential Rotation

The single largest contribution to observed solar velocity signals is generally the near solid-body rotation amounting to about 2 km/sec at the equator. The corrections due

to differential rotation with latitude are also quite substantial, exceeding 10% or 140 m/sec at 45° latitude. Following the customary format for data presentation, it is assumed that the differential rotation law at the surface is expressible as a power series in $\sin^2\lambda$:

$$\Omega = \Omega_0 (1 + A_2 \sin^2\lambda + A_4 \sin^4\lambda + \ldots) \qquad \text{(EQ 16)}$$

despite the fact that the terms in this expansion are non-orthogonal. In practice, fitting coefficients higher than the fourth order term are seldom reported, although systematic deviations from this assumed law certainly exist. The error of the assumed law near the poles appears to be less than or of the order of 20 m/sec, however (Schröter, 1985).

It is important to recognize the consistent differences in rate of rotation (and other large-scale motions) when measured by tracers and by Doppler signals. In the case of rotation, the difference is about 1%, with tracers (sunspots) rotating faster. Since the physical observable being simulated will generally be Doppler velocities, it will be appropriate to select parameters based on Doppler rotation measurements. Because the photosphere presumably exhibits some differential rotation with depth the appropriate rotation rate to use should be the Doppler rotation rate measured in the line of interest. This effect is very small, however (Pierce & LoPresto, 1984), and good observational data for the 6768Å line are lacking. Representative values for the Doppler differential rotation law, taken from measurements at Mt. Wilson in the 5250Å FeI line, are (Snodgrass, 1984; Table I):

$$\Omega_0 = 1974.1 \text{m/s}, A_2 = -0.121, A_4 = -0.178$$

This rate is somewhat slower than those generally reported (Schröter, 1985), but is probably based on the most careful analysis. It's line-of-sight component is illustrated in Figure 2, which also isolates the differential component.

3.3 Limb Shift

There are several distinct phenomena comprehended under the term limb shift. These include a displacement of the line core due to the correlation of brightness with radial velocity in unresolved convective cells (which leads to relative blue shifts at disc centre), changes in the line profile due to the variation of the brightness-averaged convective radial velocities with height in the atmosphere, and variations in the gravitational redshift (LoPresto & Pierce, 1985). The last effect is truly minusucule, going as the ratio of the difference in height of line formation to the solar radius, but the other effects can be substantial, amounting to measured variations across the disc of hundreds of m/s. The line profile variation can lead to a Doppler pulling effect, in which measured velocities are systematically reduced (or enhanced) near the limb, depending on the line sampling technique. The dependence of the limb shifts with central angle ρ is generally parametrized as an expansion in powers of $(1 - \cos\rho)$. This parametrization ignores azimuthal effects which are observed. These effect can be due to the variation of convection patterns with latitude and the line-pulling effect on basic rotation. Unfortunately, the wavelength dependence of this effect is strong and non-systematic, and little is known about the particular limb shift for the 6768Å line. Data on the line core variation taken by Pierce (reported in Jones, 1989) suggest a second-order expansion with coefficients

$a_0 = -166\text{m/s}$, $a_1 = -139\text{m/s}$, $a_2 = 700\text{m/s}$

in all directions is adequate. Such a second order expansion is assumed in the program, although data taken by Ulrich, (1991) suggest a much steeper limb dependence, and also a more significant azimuthal dependence. To the constant term should be added the virtually position-independent absolute gravitational redshift of 636 m/s (less the observer's gravitational redshift for absolute measurements; but that is generally less than 3 m/s and is ignored). This "standard" limb shift is shown in Figure 3, while Figure 4 shows the contribution to the line-of-sight velocity of the combination of the standard rotation profile and limb shift.

3.4 Meridional Flows

Observations of constant meridional velocities are ambiguous; tracer motions are of the order of a few m/s at most while Doppler measurements of components an order of magnitude larger are difficult to untangle from limb shifts (Bogart 1987). In any case, it appears that if the motions are expanded in Legendre polynomials in $\sin\lambda$ weighted by $\cos\lambda$ to avoid singularities at the poles, it is probably only the first and third order terms that contribute significantly, there being no evidence for equator crossing flows. For the sake of completeness and eventual comparison with better observational data, meridional flows are parametrized as

$$v_{\text{mer}} = \cos\lambda \sum_{i=0}^{3} a_i P_i(\sin\lambda) \qquad \text{(EQ 17)}$$

with the rather arbitrary values (in m/s) as typical:

$a_0 = -1$, $a_1 = 7.5$, $a_2 = 1$, $a_3 = 15.0$.

This "standard" meridional flow pattern is shown in Figure 5.

3.5 Stochastic Phenomena

There are four major random phenomena affecting observations of solar Doppler shifts and intensities, apart from those parametrized in the limb shift: active regions (and pores and sunspots), supergranulation, granulation, and measurement noise.

Large-scale and intense magnetic features (active regions) are not modeled at present. As their surface coverage is small even at times of maximum activity, and approaches zero at times of minimum, this is not a serious omission. The greatest interest may in fact be in modeling their effects on observables, through distortion of the integrated line profile, which are not well-understood.

Measurement noise must of course be considered, but it is quite straightforward. A normal distribution about a zero mean is simply added to the final result. The free parameter is of course the standard deviation of the noise, which must reflect instrumental characteristics. For example, MDI Doppler measurements are expected to have a per pixel noise characteristic of approximately 20 m/s.

Granulation and supergranulation are both ubiquitous small-scale velocity phenomena that affect Doppler observations. The effect of granulation on Doppler measurements is presumed to be the chief phenomenon causing limb shift. In point of fact, the

scale of granulation is smaller than the typical pixel size, so it would be very difficult to model it at the scales considered. Supergranulation, on the other hand, has a typical scale of 25 Mm (Hart., 1956), and more or less covers the solar disc uniformly (Rimmele & Schröter, 1989).

4.0 Implementation

The artificial data generator described here is implemented as a C-language function SolarView() (see Appendix I for a listing).

4.1 Dependencies

In addition to functions provided by the standrad C libary and the math library provided under Unix BSD 4.3 and similar systems, the function SolarView() makes external reference to the library function legendre_by_x() described in SOI TN-XX-XXX for the efficient calculation of the surface spherical harmonic functions used to model waves.

4.2 Calling Parameters

In order to isolate the actual generating function from any system-specific architecture designed around data handling, the library routine SolarView() operates with a parameter list that is unfortunately fairly large. The intent is that the actual call to the function with all the parameters will in general be a front-end routine designed as part of a data processing pipeline. Of course SolarView() does not operate on pre-existing data, other than its parameters, but it presumably generates data which does go through additional processing pipelines.

References

Bogart, R. S.: 1987, *Solar Phys.* **110**, 23.

Hart, A. B.: 1956, *M.N.R.A.S.* **116**, 38.

Jones, H. P.: 1989, *Solar Phys.* **120**, 211.

LoPresto, J. C. & Pierce, A. K.: 1985, *Solar Phys.* **102**, 21.

Pierce, A. K. & LoPresto, J. C.: 1984, *Solar Phys.* **93**, 155.

Pierce, A. K. & Slaughter, C. D.: 1977, *Solar Phys.* **51**, 25.

Rimmele, T. & Schröter, E. H.: 1989, *A. Ap.* **221**, 137.

Schröter, E. H.: 1985, *Solar Phys.* **100**, 141.

Snodgrass, H. B.: 1984, *Solar Phys.* **94**, 13.

Ulrich, R. K.: 1991, *Private communication.*

Appendix I SolarViewc function source code

```
/*
 *  SolarView.c                                          R S Bogart  92.09.17
 *
 *  Version 0.9
 *
 *  Generates an accurate model of projected surface spherical-harmonics (or
 *    other functions) as integrated over each pixel in a model grid placed
 *    over a spherical surface in a zenithal (near orthographic) projection.
 *    The grid is a rectangle composed of square pixels.
 *
 *  Returns number of pixels of which at least some part is on disc, i.e.,
 *    those with valid values.  A return value of 0 indicates a "serious
 *    misalignment problem" (no pixels on disc) which could be caused
 *    either by the x and/or y offset values exceeding the disc radius or
 *    by the disc radius being such a small fraction of a pixel that no
 *    quadrature sampling points happen to fall within it.
 *
 *  Planned updates:
 *
 *  1.0  Document, man page, test on multiple platforms, Makefile, remove
 *         DEBUGS
 *
 *  Required include files and libraries: * important *
 *      /usr/include/math.hfor def of trig functions
 *      /usr/local/include/Math.hfor def of Plm generators
 *
 *      /usr/lib/libm.a          for trig functions
 *      /usr/local/lib/libM.afor Plm generators
 *
 *  Calling Parameters:
 *      (double *) EffluxArray for data return; must point to a space of
 *                             at least XPixelCt * YPixelCt doubles
 *      double  MissingValValue assigned to "missing" data (pixels
 *                             completely off disc); should be NaN
 *      double  DiscRadiusApparent disc-radius, in arc-seconds; assumed
 *                             = arcsin (R_Sun / D_Obsrv_SunCtr)
 *      double  LatC
 *      double  LonC
 *      double  PixelWidth
 *      short int XPixelCtNumber of cols: pixels in x dimension
 *      short int YPixelCtNumber of rows: pixels in y dimension
 *      double  InterPixelGap
 *      double  XOffset
 *      double  YOffset
 *      double  FieldPA
 *      char    Projection
 *      (double *) LimbDarkCoeff
 *      (double *) RotationCoeff
 *      (double *) LimbShiftCoeff
 *      (double *) MeridFlowCoeff
```

```
 *       double SuperGranuleAmp
 *       int SuperGranuleK
 *       double NoiseAmp
 *       (MODE *) Mode
 *       int      ModeCt
 */

#include <math.h>
#include <Math.h>

#define VERS_NUM(0.9)
#define INTENS   (0)
#define VLOS     (1)
#define VTRANS   (2)

typedef struct mode_struct {
  int l;
  int m;
  float amp;
  float phase;
  struct mode_struct *next;
} MODE;

struct sample_grid {
  double x;
  double y;
  double w;
  double proj[5];
  double sinlat;
  double lon;
  double rot;
  double shift;
  double merid;
  double up;
  double val;
} *setup_sampling (char, int *);

double random_normal (double, double);

double ang_r, sinang_r, tanang_r, sinlatc, coslatc;

int SolarView  (double *Efflux,
                double MissingVal,
                double DiscRadius,
                double LatC, double LonC,
                double PixelWidth,
                short int XPixelCt, short int YPixelCt,
                double InterPixelGap,
                double XOffset, double YOffset,
                double FieldPA,
                char Projection,
                double LimbDarkCoeff[5],
```

```
                double RotationCoeff[3],
                double LimbShiftCoeff[3],
                double MeridFlowCoeff[4],
                double SuperGranuleAmp, int SuperGranuleK,
/*              double GranuleAmp, int GranuleK,*/
                double NoiseAmp,
                MODE *Mode,
                int ModeCt)
{
struct sample_grid *smp, *pxl_smp;
double h, invrs_width, r, r2, r2_include, scaled_radius, sum_wt, v, vc;
double xc, yc, xcp, ycp, rc2, sig, chi, coslat, sinlat, sin2lat, lat, lon;
double mu, sinth, logmu, z, limbd;
double cospa, sinpa, scale_fac;
double rotation, rot_base_rate, limb_shift, merid_flow;
double supergran_period, supergran_merid, supergran_zonal, granule;
double x_phase, y_phase, rsg, asg, vsg;
double plm[4];
double cvg_diff;
float output_val;
int good_pxl_ct, pxl_ct, sample_pts, sample_ctr;
int i, n, modes, observ;
short int row, col;

sinlatc = sin (LatC);
coslatc = cos (LatC);
cospa = cos (FieldPA);
sinpa = sin (FieldPA);

observ = (Projection == 0) ? INTENS : (Projection == 4) ? VTRANS : VLOS;

good_pxl_ct = pxl_ct = 0;
/*
 *  ang_r : apparent solar disc radius [radian]
 *  scaled_radius : image disc radius in units of half field width
 *  r2_include : square of maximum radius on image plane at which any part of
 *    pixel is on disc, in units of scaled_radius
 *  h : half width of effective square on pixel, in units of scaled_radius
 */
ang_r = atan (1.0) * DiscRadius / 3600.0 / 45.0;
sinang_r = sin (ang_r);
tanang_r = tan (ang_r);
invrs_width = (XPixelCt > YPixelCt) ? 1.0 / XPixelCt : 1.0 / YPixelCt;
scaled_radius = tanang_r * invrs_width /
                  tan (atan (1.0) * PixelWidth / 3600.0 / 90.0);
r2_include = 1.0 + 2.0 * sqrt (2.0) * invrs_width  / scaled_radius;
r2_include *= r2_include;
h = (1.0 - InterPixelGap) * invrs_width  / scaled_radius;
smp = setup_sampling (2, &sample_pts);
sample_ctr = 4;
pxl_smp = (struct sample_grid *) malloc (sample_pts *
                                    sizeof (struct sample_grid));
```

```
scale_fac = 1.0 * invrs_width / scaled_radius;
for (row=0; row<YPixelCt; row++)
{
  ycp = (2*row - YPixelCt + 1 - 2*YOffset) * scale_fac;
  for (col=0; col<XPixelCt; col++)
  {
#ifdef DEBUG
    printf ("[%d, %d]:\n", col, row);
#endif
    xcp = (2*col - XPixelCt + 1 - 2*XOffset) * scale_fac;
    xc = xcp * cospa - ycp * sinpa;
    yc = ycp * cospa + xcp * sinpa;
    rc2 = xc * xc + yc * yc;
    if (rc2 < r2_include)
    {
      sum_wt = v = vc = 0.0;
      for (n=0; n<sample_pts; n++)
      {
        pxl_smp->x = xc + h * smp->x;
        pxl_smp->y = yc + h * smp->y;
        r2 = pxl_smp->x * pxl_smp->x + pxl_smp->y * pxl_smp->y;
        if (r2 >= 1.0)
        {
          pxl_smp->w = 0.0;
          pxl_smp->sinlat = pxl_smp->lon = 0.0;
        }
        else
        {
          pxl_smp->w = smp->w;
#ifdef DEBUG
          printf ("    [%7.4f %7.4f] : ", pxl_smp->x, pxl_smp->y);
#endif
          map (pxl_smp->x, pxl_smp->y, r2, &r, &lon, &lat, &sinlat, &coslat,
               &sig, &chi, &mu, &sinth);
/*
 *  Calculate all projections, as the two most complicated (2 & 3) are always
 *    needed as long as rotation and meridional flows are included, unless
 *    observable is intensity
 */
                                        /*  Scalar quantity (no projection)  */
          pxl_smp->proj[0] = 1.0;
          if (observ != INTENS)
          {
                  /*  Line-of-sight component of radial surface component  */
            pxl_smp->proj[1] = -mu;
               /*  Line-of-sight component of meridional surface component  */
            pxl_smp->proj[2] = (coslat > 1.e-6) ?
                sinth * (coslatc * cos (r) * cos (chi) -
                                sinlatc * sin (r)) / coslat : 0.0;
                 /*  Line-of-sight component of zonal surface component  */
            pxl_smp->proj[3] = (coslat > 1.e-6) ?
```

```
                sinth * coslatc * sin (chi) / coslat : 0.0;
 /*  Transverse component (toward disc centre) of radial surface component  */
            pxl_smp->proj[4] = -sinth;
          }
/*
 *  Weight by limb-darkening function : use expansion in powers of log mu
 */
          logmu = log (mu);
          z = limbd = 1.0;
          for (i=0; i<5; i++)
          {
            z *= logmu;
            limbd += LimbDarkCoeff[i] * z;
          }
          pxl_smp->w *= limbd;
          pxl_smp->sinlat = sinlat;
          pxl_smp->lon = lon;

          if (observ == INTENS)
            pxl_smp->proj[Projection] *= limbd;
          else
          {
/*
 *  Add differential rotation: classical expansion in even powers of sin lat
 */
            z = coslat;
            rotation = rot_base_rate = RotationCoeff[0] * z;
            z = sin2lat = sinlat * sinlat;
            for (i=1; i<3; i++)
            {
              rotation += rot_base_rate * RotationCoeff[i] * z;
              z *= sin2lat;
            }
            pxl_smp->rot = rotation;
/*
 *  Add limb shift: expansion in powers of (1 - mu)
 */
            limb_shift = 0.0;
            z = 1.0;
            for (i=0; i<3; i++)
            {
              limb_shift += LimbShiftCoeff[i] * z;
              z *= 1.0 - mu;
            }
            pxl_smp->shift = limb_shift;
/*
 *  Add meridional flow: expansion in Legendre polynomials in sin lat
 */
            merid_flow = 0.0;
            legendre_by_l (plm, sinlat, 3, 0, 1);
            for (i=0; i<4; i++)
            {
```

```
            merid_flow += plm[i] * MeridFlowCoeff[i] * coslat;
          }
          pxl_smp->merid = merid_flow;
          pxl_smp->up = 0.0;
/*
 *  Add "supergranulation": periodic functions in lat and lon*coslat for
 *    zonal and meridional flows
 */
          if (SuperGranuleAmp != 0.0)
          {
            x_phase = fmod ((lon - LonC) * 2 * SuperGranuleK  * coslat, 2.0);
            y_phase = fmod (lat * 2 * SuperGranuleK, 2.0);
            if (x_phase > 1.0)
              x_phase -= 2.0;
            if (y_phase > 1.0)
              y_phase -= 2.0;
            if (x_phase < -1.0)
              x_phase += 2.0;
            if (y_phase < -1.0)
              y_phase += 2.0;
            rsg = hypot (x_phase, y_phase);
            asg = atan2 (y_phase, x_phase);
            vsg = SuperGranuleAmp * sin (rsg * M_PI);
            if (vsg > 0.0)
            {
              supergran_zonal = vsg * cos (asg);
              supergran_merid = vsg * sin (asg);
              pxl_smp->rot += supergran_zonal;
              pxl_smp->merid += supergran_merid;
            }
          }
/*
 *  Add "granulation": periodic function in lat and lon*coslat for
 *    radial flow
 */
/*
          if (GranuleAmp != 0.0)
          {
            x_phase = fmod (lon * 2 * GranuleK  * coslat, 2.0);
            y_phase = fmod (lat * 2 * GranuleK, 2.0);
            if (x_phase > 1.0)
              x_phase -= 2.0;
            if (y_phase > 1.0)
              y_phase -= 2.0;
            if (x_phase < -1.0)
              x_phase += 2.0;
            if (y_phase < -1.0)
              y_phase += 2.0;
            rsg = hypot (x_phase, y_phase);
            if (rsg < 1.0)
              pxl_smp->up += cos (rsg * M_PI_2);
          }
```

```
*/
          }
        }
        sum_wt += pxl_smp->w;
        pxl_smp++;
        smp++;
      }
      smp -= sample_pts;
      pxl_smp -= sample_pts;
      modes = ModeCt;
      while (modes--)
      {
        Ylm_by_x (pxl_smp, sample_pts, Mode->l, Mode->m,
                  Mode->phase);
        for (n=0; n<sample_pts; n++)
        {
          v += pxl_smp[n].w * Mode->amp * pxl_smp[n].proj[Projection] *
               pxl_smp[n].val;
        }
        n = sample_ctr;
        if (pxl_smp[n].w > 0.0)
          vc += Mode->amp * pxl_smp[n].proj[Projection] * pxl_smp[n].val;
        Mode = Mode->next;
      }
      if (observ == VLOS)
      {
        for (n=0; n<sample_pts; n++)
        {
          v += pxl_smp[n].w * pxl_smp[n].rot * pxl_smp[n].proj[3];
          v += pxl_smp[n].w * pxl_smp[n].merid * pxl_smp[n].proj[2];
          v += pxl_smp[n].w * pxl_smp[n].up * pxl_smp[n].proj[1];
          v += pxl_smp[n].w * pxl_smp[n].shift;
        }
        n = sample_ctr;
        if (pxl_smp[n].w > 0.0)
        {
          vc += pxl_smp[n].rot * pxl_smp[n].proj[3];
          vc += pxl_smp[n].merid * pxl_smp[n].proj[2];
          vc += pxl_smp[n].up * pxl_smp[n].proj[1];
          vc += pxl_smp[n].shift;
        }
      }
      else if (observ == INTENS)
      {
        for (n=0; n<sample_pts; n++)
        {
          v += pxl_smp[n].w * pxl_smp[n].proj[0];
        }
        n = sample_ctr;
        if (pxl_smp[n].w > 0.0)
        {
          vc += pxl_smp[n].proj[0];
```

```
        }
      }
      if (sum_wt > 0.0)
      {
        v /= sum_wt;
        good_pxl_ct++;
        if (pxl_smp[sample_ctr].w > 0.0)
          cvg_diff = fabs (vc - v);
#ifdef DEBUG
        printf ("\t%12.5e\n", v);
#endif
      }
      else
      {
#ifdef DEBUG
        printf ("************\n");
#endif
        v = MissingVal;
      }
    }
    else
    {
#ifdef DEBUG
      printf ("[******* *******] : ************\n");
#endif
      v = MissingVal;
    }
    if (NoiseAmp != 0.0)
      v += random_normal (0.0, NoiseAmp);
    Efflux[pxl_ct++] = v;
  }
}
return (good_pxl_ct);
}

map (x, y, r2, rho, lon, lat, sinlat, coslat, sig, pa, mu, sinth)
double x, y, r2, *lon, *lat, *sinlat, *coslat, *rho, *sig, *pa, *mu, *sinth;
/*
 *  Map projected coordinates (x, y) to (lon, sin (lat)) and (rho | sig, pa)
 *
 *  rho is the angle point:sun_center:observer
 *  sig is the angle point:observer:sun_center
 *  mu is cosine of angle between the point:observer line and the local normal
 *
 *  ang_r is the angular radius of the sun at the observer's tangent line
 *  latc is the heliographic latitude of disc centre, uncorrected for light
 *    travel time
 */
{
double r, sinr, sinlon, sinsig;
extern double ang_r, sinang_r, tanang_r, sinlatc, coslatc;
```

```
r = sqrt (r2);
            /*  Camera curvature correction, no small angle approximations  */
*sig = atan (r * tanang_r);
sinsig = sin (*sig);
*rho = asin (sinsig / sinang_r) - *sig;
sinr = sin (*rho);
*mu = cos (*rho + *sig);
*sinth = sin (*rho + *sig);
*pa = atan2 (x , y);
*sinlat = sinlatc * cos (*rho) + coslatc * sinr * cos (*pa);
*coslat = sqrt (1.0 - *sinlat * *sinlat);
sinlon = sinr * sin (*pa) / *coslat;
*lon = asin (sinlon);
*lat = asin (*sinlat);
}

struct sample_grid *setup_sampling (char type, int *sample_pts)
{
double d;
int n;
struct sample_grid *smp, *smp0;

switch (type)
  {
  case 1:
    n = *sample_pts = 4;
    break;
  case 2:
  case 3:
    n = *sample_pts = 9;
    break;
  default:
    n = *sample_pts = 1;
  }
smp = smp0 = (struct sample_grid *)malloc (*sample_pts *
                                    sizeof (struct sample_grid));
switch (type)
  {
  case 1:
    d = sqrt (1. / 3.);
    smp->x = smp->y = -d;
    smp++;
    smp->x = d; smp->y = -d;
    smp++;
    smp->x = -d; smp->y = d;
    smp++;
    smp->x = smp->y = d;
    smp = smp0;
    while (n--)
      {
        smp->w = 0.25;
        smp++;
```

```
      }
    break;
  case 2:
    d = 1.0;
    smp->x = smp->y = -d; smp->w = 1.0 / 36.0;
    smp++;
    smp->x = 0.0; smp->y = -d; smp->w = 1.0 / 9.0;
    smp++;
    smp->x = d; smp->y = -d; smp->w = 1.0 / 36.0;
    smp++;
    smp->x = -d; smp->y = 0.0; smp->w = 1.0 / 9.0;
    smp++;
    smp->x = smp->y = 0.0; smp->w = 4.0 / 9.0;
    smp++;
    smp->x = d; smp->y = 0.0; smp->w = 1.0 / 9.0;
    smp++;
    smp->x = -d; smp->y = d; smp->w = 1.0 / 36.0;
    smp++;
    smp->x = 0.0; smp->y = d; smp->w = 1.0 / 9.0;
    smp++;
    smp->x = smp->y = d; smp->w = 1.0 / 36.0;
    smp = smp0;
    break;
  case 3:
    d = sqrt (0.6);
    smp->x = smp->y = -d; smp->w = 25.0 / 324.0;
    smp++;
    smp->x = 0.0; smp->y = -d; smp->w = 10.0 / 81.0;
    smp++;
    smp->x = d; smp->y = -d; smp->w = 25.0 / 324.0;
    smp++;
    smp->x = -d; smp->y = 0.0; smp->w = 10.0 / 81.0;
    smp++;
    smp->x = smp->y = 0.0; smp->w = 16.0 / 81.0;
    smp++;
    smp->x = d; smp->y = 0.0; smp->w = 10.0 / 81.0;
    smp++;
    smp->x = -d; smp->y = d; smp->w = 25.0 / 324.0;
    smp++;
    smp->x = 0.0; smp->y = d; smp->w = 10.0 / 81.0;
    smp++;
    smp->x = smp->y = d; smp->w = 25.0 / 324.0;
    smp = smp0;
    break;
  default:
    smp->x = smp->y = 0.0;
    smp->w = 1.0;
  }
return (smp0);
}

int Ylm_by_x (pxl, nx, l, m, phase)
```

```
struct sample_grid *pxl;
int nx, l, m;
float phase;
{
static int allocd = 0;
static double *x, *ylm;
int n, status;

if (!allocd)
{
  x = (double *)malloc (nx * sizeof (double));
  ylm = (double *)malloc (nx * sizeof (double));
  allocd = nx;
}
if (allocd < nx)
{
  x = (double *)realloc (x, nx * sizeof (double));
  ylm = (double *)realloc (ylm, nx * sizeof (double));
  allocd = nx;
}

for (n=0; n<nx; n++)
{
  x[n] = pxl[n].sinlat;
}
status = legendre_by_x (ylm, x, nx, l, m, 1);
for (n=0; n<nx; n++)
{
  pxl[n].val = ylm[n];
  if (m)
    pxl[n].val *= cos (m * (pxl[n].lon - phase));
}

return (status);
}

#define MXINT(0x7fffffff)

double random_normal (double mean, double stdv)
{
static int calls;
static double stp, x1, x2, y;
double ru;

if (!calls)
/*  On first call, initialize random number generator and constants  */
{
  int seed;
  seed = getpid ();
  srandom (seed);
  stp = 2.0 / MXINT;
}
```

```
calls++;
if (calls & 1)
{
  ru = 2.0;
  while (ru >= 1.0)
  {
    x1 = stp * random () - 1.0;
    x2 = stp * random () - 1.0;
    ru = x1 * x1 + x2 * x2;
  }
  y = stdv * sqrt (-2.0 * log (ru) / ru);
  return (y * x1 + mean);
}
else
{
  return (y * x2 + mean);
}
}
```