This file consists of Chapter 5 of **MacAnova User's Guide** by Gary W. Oehlert and Christopher Bingham, issued as Technical Report Number 617, School of Statistics, University of Minnesota, revised August 1998, describing Version 4.07 of MacAnova.

Fonts used in this manual are Palatino, Courier, and Symbol.

For information concerning MacAnova, write University of Minnesota, Department of Applied Statistics, 352 Classroom Office Building, 1994 Buford Avenue, St. Paul, MN 55108-6042.

5. Time Series related functions

5.1 **Introduction** MacAnova has a suite of functions useful in the frequency analysis of univariate and multivariate time series. Besides the basic Fourier transform functions, `rft()`, `hft()`, and `cft()`, there are functions to compute discrete convolutions and sums of lagged products and to manipulate complex (real + *i**imaginary) data. `autoreg()` and `movavg()` can be used to generate data from ARIMA models, or to compute innovations from ARIMA data when the coefficients in the model are known. `yulewalker()` and `partacf()` allow computing autoregression coefficients and partial autocorrelation functions from autocorrelation functions and vice versa. `toeplitz()` transforms an autocovariance function into a variance/covariance matrix.

Distributed with MacAnova is a file, `Tser.mac`, which contains macros that are useful in time series analysis and which illustrate the use of some of the functions described in this chapter. The most important of these macros are described in Sec. 5.4. File `Tser.hlp` contains help on these macros. See Sec. 8.6.1 for information on how to use an alternate help file.

5.2 **Operations useful in frequency domain time series analysis** Although MacAnova has no built in commands specifically for spectrum and cross-spectrum analyses of time series, its functions for computing discrete Fourier transforms and for working with complex data allow quite sophisticated frequency domain analyses, either directly or by using macros. These are the basis of the macros in `Tser.mac`.

Here are some conventions used in this chapter:

> The uncapitalized words “real”, “imaginary”, and “complex” are used in their mathematical sense.
>
> The capitalized words “Real” and “Complex” are used to describe a MacAnova variable when it is an argument to or output from one of the functions described in this section. See Sec. 5.2.3.
>
> REAL is used when referring to the type of a MacAnova variable.

For example, we will write that 3 + 4*i* is a complex number, that `cmplx()` takes two Real arguments and returns a Complex result, and that a *n* by *p* Complex matrix is represented by a REAL matrix with 2*p* columns.

5.2.1 **The DFT (Discrete Fourier Transform)** Let $\{X_t\} = \{X_0, X_1, ..., X_{N-1}\}$ be a finite complex or real series, that is, a series of N complex or real numbers. Then the DFT of $\{X_t\}$ is the finite complex series of length N

$$\{\hat{X}_k\}, \text{ where } \hat{X}_k = \sum_{t=0}^{N-1} X_t e^{-i\,2\pi tk\,/\,N},\ k = 0, 1, ..., N-1.$$

We will also use the notation $\text{DFT}\{X\}_k = \hat{X}_k$ and will often omit the qualifier “finite” and refer simply to complex and real series.

Note that in the definition of the DFT, indices start with 0 rather than 1 as is assumed

in MacAnova. In working with Fourier transforms in MacAnova, this correspondence must be kept in mind. The first element in a series is always considered to be X_0 or $\hat{X}_0$, but to access it you will need to use `x[1]`.

If $\{\hat{X}_k\}_{k=0}^{N-1}$ is the DFT of $\{X_t\}_{t=0}^{N-1}$, then the following *inversion formula* holds:

$$X_t = \frac{1}{N}\sum_{k=0}^{N-1}\hat{X}_k e^{+i2\pi tk/N} = N^{-1}\overline{\mathrm{DFT}}\{\overline{\mathrm{DFT}\{X\}}\}_t,\ t = 0, 1, ..., N-1,$$

where $\bar{z}$ is the complex conjugate of the complex number z.

When dealing with the DFT it is sometimes convenient to consider a finite complex or real series as a segment of length N from a *periodic* infinite complex or real series $\{X_t^{(N)}\}_{t=-\infty}^{\infty}$ with period N. With this in mind, we define

$$X_t^{(N)} = X_t,\ t = 0, 1, ..., N-1,\ X_{N+s}^{(N)} = X_s^{(N)},\ s = 0, 1, ... \text{ and } X_{-s}^{(N)} = X_{-s+N}^{(N)},\ s = 1, 2, ...$$

With this extended definition, it can be verified that, if t_0 is an arbitrary integer, DFT$\{X\}$ can be expressed as

$$\hat{X}_k = \sum_{t=t_o}^{N+t_0-1} X_t^{(N)} e^{-i2\pi tk/N},\ k = 0, \pm1, \pm2, ...$$

that is, as a summation over an arbitrary complete period of $X_t^{(N)}$.

More generally, we can define a periodic extension $\{X_t^{(S)}\}$ with period $S \geq N$

$$X_t^{(S)} = X_t,\ t = 0, 1, ..., N-1,\ X_t^{(S)} = 0,\ s=N,...,S-1,$$
$$X_{S+s}^{(S)} = X_s^{(S)},\ s = 0, 1, ... \text{ and } X_{-s}^{(S)} = X_{-s+S}^{(S)},\ s = 1, 2, ...$$

We notate the DFT of $\{X_t^{(S)}\}$ as $\hat{X}_k^{(S)} = \sum_{t=0}^{S-1} X_t^{(S)} e^{-i2\pi tk/S}$, $k = 0, ..., S-1$. This definition makes sense for an arbitrary integer k, and thus can be taken to define an infinite sequence with period S. With this notation, $\hat{X}_k = \hat{X}_k^{(N)}$.

A complex series $\{Y_j\}$ of length N that satisfies

$$Y_0 \text{ is real and } Y_{N-j} = \overline{Y}_j,\ j = 1, ..., N-1$$

is said to have *Hermitian symmetry*, or simply to be a *Hermitian* series.

This definition implies that when N is even, $Y_{N/2}$ is real. Viewed as a segment of an infinite periodic complex series $\{Y_j^{(N)}\}$, a Hermitian series satisfies $Y_{-j}^{(N)} = \overline{Y_j^{(N)}}$, $j = 0, 1, ..., .$ When $\{X_t\}_{t=0}^{N-1}$ is a real series, then its DFT, $\{\hat{X}_k\}_{k=0}^{N-1}$, is a Hermitian series. Conversely, when $\{X_t\}_{t=0}^{N-1}$ is Hermitian, $\{\hat{X}_k\}_{k=0}^{N-1}$ is real.

If $\{X_j\}_{j=0}^{N-1}$ is an unrestricted complex series, its *Hermitian symmetrized form* is the Hermitian series $\{\tilde{X}_j\}_{j=0}^{N-1}$ where

$$\tilde{X}_0 = \mathrm{Re}(X_0),\ \sim\tilde{X}_j = (1/2)(X_j + \overline{X_{N-j}}),\ j = 1, \ldots, N-1$$

As an infinite periodic series, this can be equivalently written

$$\tilde{X}_j = (1/2)(X_j + \overline{X_{-j}}),\ j = 0, \pm1, \pm2, \ldots$$

5.2.2 **Continuous Fourier transform of a finite series** The *continuous Fourier transform* of a finite real or complex series $\{X_t\}_{t=0}^{N-1}$ is the continuous function of the real variable f

$$\hat{X}(f) = \mathrm{CFT}\{X\}(f) = \sum_{t=0}^{N-1} X_t e^{-i\,2\pi t f}$$

The argument f is the *frequency* at which $\hat{X}(f)$ is evaluated and is in units of cycles. $\hat{X}(f)$ is a periodic function of f with period 1, that is $\hat{X}(f \pm k) = \hat{X}(f)$ for any integer k. You can consider the DFT to be a "sampling" of the CFT, in the sense that $\hat{X}_k = \hat{X}(k/N)$.

Define the finite series $\{X_t^{\#}\}$ of length $S > N$ by $X_t^{\#} = X_t^{(S)}$, $t = 1, \ldots, N-1$, $X_t^{\#} = 0$, $t = N, \ldots, S-1$, that is, $\{X_t\}$ padded with $S-N$ zeros. Then $\hat{X}_k^{\#} = \hat{X}_k^{(S)} = \hat{X}(k/S)$. Thus, by padding $\{X_t\}$ by additional zeros, you can use the DFT to compute the CFT at a denser set of frequencies.

5.2.3 **Representation of Real, Hermitian and Complex Series** Real series $\{X_t\}_{t=0}^{N-1}$ are stored as *columns* of REAL matrices with N rows, with X_0 in row 1.

Hermitian series of length N are also stored as columns of REAL matrices with N rows. This is possible because a Hermitian series $\{X_j\}$ of length N is fully determined by N real numbers, X_0, $\mathrm{Re}(X_1)$, ..., $\mathrm{Re}(X_{\lfloor (N-1)/2 \rfloor})$, $\mathrm{Im}(X_1)$, ..., $\mathrm{Im}(X_{\lfloor (N-1)/2 \rfloor})$, and, if N is even, $X_{N/2}$. Here the notation $\lfloor \alpha \rfloor$ means the largest integer not greater than α, and corresponds to the `floor()` function in MacAnova. These N numbers are stored in a column of a REAL matrix in the somewhat peculiar order

$$X_0, \mathrm{Re}(X_1), \ldots, \mathrm{Re}(X_{\lfloor (N-1)/2 \rfloor}), \{X_{N/2}\}, \mathrm{Im}(X_{\lfloor (N-1)/2 \rfloor}), \ldots, \mathrm{Im}(X_2), \mathrm{Im}(X_1),$$

where $X_{N/2}$ is included only when N is even. That is, the real parts of the elements in the first half of the series come first, followed by their imaginary parts, *in reverse order*, omitting the one or two imaginary parts, $\mathrm{Im}(X_0)$ and $\mathrm{Im}(X_{N/2})$, known to be zero .

Unrestricted Complex series are stored in adjacent pairs of columns of REAL matrices, the real parts in odd numbered columns and the imaginary parts in even numbered columns. Thus a n by p complex matrix is represented by a n by $2p$ REAL MacAnova matrix, with the real and imaginary parts of column j of the complex matrix in columns $2j-1$ and $2j$ of the MacAnova matrix, $j = 1, \ldots, p$.

Examples:

```
Cmd> print(hx,hy) # two matrices, each containing 2 Hermitian series
hx:
(1,1)            1            2
(2,1)            3            4
(3,1)            5            6
(4,1)            7            8
(5,1)            9           10
hy:
(1,1)            1            2
(2,1)            3            4
(3,1)            5            6
(4,1)            7            8
(5,1)            9           10
(6,1)           11           12
```

When considered as containing two Hermitian series, matrix hx represents the 5 by 2 complex matrix $\begin{bmatrix} 1+0i & 2+0i \\ 3+9i & 4+10i \\ 5+7i & 6+8i \\ 5-7i & 6-8i \\ 3-9i & 4-10i \end{bmatrix}$, and matrix hy represents the 6 by 2 complex matrix

$$\begin{bmatrix} 1+0i & 2+0i \\ 3+11i & 4+12i \\ 5+9i & 6+10i \\ 7+0i & 8-0i \\ 5-9i & 6-10i \\ 3-11i & 4-12i \end{bmatrix}.$$

```
Cmd> print(cx) # Complex matrix containing 2 complex series
cx:
(1,1)            1            2            3            4
(2,1)            5            6            7            8
(3,1)            9           10           11           12
(4,1)           13           14           15           16
(5,1)           17           18           19           20
```

Matrix cx represents the 5 by 2 complex matrix $\begin{bmatrix} 1+2i & 3+4i \\ 5+6i & 7+8i \\ 9+10i & 11+12i \\ 13+14i & 15+16i \\ 17+18i & 19+20i \end{bmatrix}$.

If a matrix with an *odd* number of columns is an argument to any function expecting a Complex series, it is considered to have adjoined an extra column consisting of zeros, that is the last column of the Complex matrix is considered to be real. When a *m* by *n* Complex matrix (*m* by 2*n* or *m* by 2*n*–1 REAL matrix) is an argument to function

`cft()`, rows 1, 2, ..., m correspond to first subscript 0, 1, ..., m –1.

Let $z = x + iy$ be a complex number. Then x and y are its *rectangular* or Cartesian coordinates. Its *polar* coordinates are its *amplitude* or modulus $r = |z| = \sqrt{x^2 + y^2}$, and its *phase* or argument $\varphi = \arg(z)$, where φ satisfies $x = r\cos\varphi$ and $y = r\sin\varphi$. Thus $z = r(\cos\varphi + i \sin\varphi) = re^{i\varphi}$. One way of representing the polar form of $re^{i\varphi}$ is as the complex number $r + i\varphi$. If $X_0, X_1, ..., X_{N-1}$ is a Hermitian series, thaen, provided r_0 is defined to be X_0, the series $r_0, r_1 + i\varphi_1, r_2 + i\varphi_2, ..., , r_{N-1} + i\varphi_{N-1}$ is also Hermitian sinc $r_k = r_{N-k}$ and $\varphi_k = -\varphi_{N-k}$. By an abuse of notation we define $r_0 = X_0$, and $r_{m/2} = X_{m/2}$ to be the "amplitudes" of the real elements of a Hermitian series, even though they may be negative. This representation is exploited by `hpolar()` and `cpolar()` (Sec. 5.2.4).

A Complex series of length m, can be specified in rectangular form by exactly $2m$ real numbers. Because of the representation of the polar form as $r + i\varphi$ this is also true for its polar form. Similarly, both the rectangular and polar forms of a Hermitian series $\{X_k\}$ of length m require m real numbers since the polar form has a Hermitian complex representation. For this reason, MacAnova stores the polar forms of both Hermitian and Complex matrices as matrices of the same type, Hermitian or Complex, with real parts $\{r_k\}$ and imaginary parts $\{\varphi_k\}$. Functions `hpolar()`, `cpolar()`, `hrect()`, and `crect()` allow conversion from rectangular form to polar form and vice versa.

Note that these conventions concern only the *interpretation* of the contents of certain REAL MacAnova matrices and vectors. No internal record is maintained as to whether a matrix is Real, Hermitian, or Complex. In particular, many functions such as `solve()` and `diag()`, and operations such as `*` and `%*%` yield nonsense results when applied to matrices whose columns are considered to be Hermitian or Complex series. Elementwise addition and subtraction, `+` and `-`, give correct results as long as dimensions match.

In examples, variable names such as `rx` and `ry` will represent matrices whose columns are considered to be Real series; `hx` and `hy` will represent matrices whose columns are considered to be Hermitian series; and `cx` and `cy` will represent matrices whose columns are considered in pairs to be unrestricted Complex series. Also, by an abuse of terminology we sometimes will use "Real series", "Hermitian series", or "Complex series" to refer to entire matrices whose columns represent such series.

5.2.4 **Functions for manipulating Complex and Hermitian series – hconj(), cconj(), hreal(), creal(), himag(), cimag(), cpolar(), hpolar(), crect(), hrect(), htoc(), ctoh(), cmplx()**

Most of these operations come in two "flavors", one whose name starts with "h" expecting a Hermitian series as argument, and one whose name starts with "c" expecting a Complex series.

`hconj()` and `cconj()` compute the complex conjugates of their arguments.

```
Cmd> hxj <- hconj(hx);cxj <- cconj(cx);print(hxj,cxj)
hxj:
(1,1)            1            2
(2,1)            3            4
(3,1)            5            6
(4,1)           -7           -8
(5,1)           -9          -10
cxj:
(1,1)            1           -2            3           -4
(2,1)            5           -6            7           -8
(3,1)            9          -10           11          -12
(4,1)           13          -14           15          -16
(5,1)           17          -18           19          -20
```

`hreal()` and `creal()` return the real parts of their arguments as Real series. Similarly, `himag()` and `cimag()` return the imaginary parts of their arguments, also as Real series.

```
Cmd> print(REhx:hreal(hx),IMhx:himag(hx)) # Note the symmetries
REhx:
(1,1)            1            2
(2,1)            3            4
(3,1)            5            6
(4,1)            5            6
(5,1)            3            4
IMhx:
(1,1)            0            0
(2,1)            9           10
(3,1)            7            8
(4,1)           -7           -8
(5,1)           -9          -10
Cmd> print(REcx:creal(cx),IMcx:cimag(cx))
REcx:
(1,1)            1            3
(2,1)            5            7
(3,1)            9           11
(4,1)           13           15
(5,1)           17           19
IMcx:
(1,1)            2            4
(2,1)            6            8
(3,1)           10           12
(4,1)           14           16
(5,1)           18           20
```

You can use `htoc()` (**H**ermitian **to C**omplex) and `ctoh()` (**C**omplex **to H**ermitian) to translate between the two forms. `htoc(hx)` returns the Complex form of the Hermitian series `hx`, while `ctoh(cx)` returns the Hermitian symmetrized form of the Complex series `cx`. The composition `ctoh(htoc(hx))` amounts to the identity transformation, but `htoc(ctoh(cx))` does not, unless `cx` has Hermitian symmetry to start with.

```
Cmd> htoc(hx)
(1,1)            1            0            2            0
(2,1)            3            9            4           10
(3,1)            5            7            6            8
(4,1)            5           -7            6           -8
(5,1)            3           -9            4          -10

Cmd> ctoh(cx)
(1,1)            1            3
(2,1)           11           13
(3,1)           11           13
(4,1)           -2           -2
(5,1)           -6           -6
```

cmplx() creates a Complex series from one or two Real series.

```
Cmd> cmplx(creal(cx),cimag(cx)) # same as cx
(1,1)            1            2            3            4
(2,1)            5            6            7            8
(3,1)            9           10           11           12
(4,1)           13           14           15           16
(5,1)           17           18           19           20

Cmd> cmplx(creal(cx)) # missing 2nd arg assumed 0
(1,1)            1            0            3            0
(2,1)            5            0            7            0
(3,1)            9            0           11            0
(4,1)           13            0           15            0
(5,1)           17            0           19            0
```

hpolar() and cpolar() transform Hermitian or Complex series in rectangular form into polar form (Sec. 5.2.3). hrect() and crect() are inverses to hpolar() and cpolar(), transforming from polar form to rectangular form.

```
Cmd> print(hpolar:hpolar(hx),cpolar:cpolar(cx))
hpolar:
(1,1)            1            2
(2,1)       9.4868        10.77
(3,1)       8.6023           10
(4,1)      0.95055       0.9273
(5,1)        1.249       1.1903
cpolar:
(1,1)       2.2361       1.1071            5       0.9273
(2,1)       7.8102      0.87606        10.63      0.85197
(3,1)       13.454      0.83798       16.279      0.82885
(4,1)       19.105      0.82242       21.932      0.81765
(5,1)       24.759      0.81396       27.586      0.81103

Cmd> # Use hypot() & atan() (Sec. 2.8.6 ) to confirm result

Cmd> hypot(cx[,vector(1,3)],cx[,vector(2,4)]) # = creal(polar(cx))
(1,1)       2.2361            5
(2,1)       7.8102        10.63
(3,1)       13.454       16.279
(4,1)       19.105       21.932
(5,1)       24.759       27.586
```

```
Cmd> atan(cx[,vector(2,4)],cx[,vector(1,3)]) # = cimag(polar(cx))
(1,1)       1.1071       0.9273
(2,1)      0.87606      0.85197
(3,1)      0.83798      0.82885
(4,1)      0.82242      0.81765
(5,1)      0.81396      0.81103

Cmd> print(hrect:hrect(hpolar(hx)),crect:crect(cpolar(cx)))
hrect:
(1,1)            1            2   Same as hx
(2,1)            3            4
(3,1)            5            6
(4,1)            7            8
(5,1)            9           10
crect:
(1,1)            1            2            3            4 Same as
(2,1)            5            6            7            8 cx
(3,1)            9           10           11           12
(4,1)           13           14           15           16
(5,1)           17           18           19           20
```

The φ's in the "imaginary" parts of the output of `hpolar()` and `cpolar()` (see Sec. 5.2.4) are in the units specified by option `angles` as set by `setoptions()` (see Sec. 8.1.3). Possible values for `angles` are `"radians"`, `"degrees"`, and `"cycles"`, with `"radians"` being the default. Conversely the "imaginary" parts of arguments to `hrect()` and `crect()` are assumed to be in these same units. In the example, angles are in radians. However, although it takes a bit of getting used to, in the frequency analysis of time series, measuring angles in *cycles* (360° = 1 cycle) is often the most convenient.

```
Cmd> setoptions(angles:"cycles") # angles assumed in cycles

Cmd> cpolar(cx)
(1,1)       2.2361      0.17621            5      0.14758
(2,1)       7.8102      0.13943        10.63      0.13559
(3,1)       13.454      0.13337       16.279      0.13192
(4,1)       19.105      0.13089       21.932      0.13013
(5,1)       24.759      0.12955       27.586      0.12908
```

A complicating fact is that $\varphi = \arg(z)$ is not uniquely defined, in that, when $z = r(\cos\varphi + i \sin\varphi)$, then also $z = r\{\cos(\varphi \pm 2k\pi) + i \sin(\varphi \pm 2k\pi)\}$ for any integer k. The usual mathematical definition obtains uniqueness by imposing some simple condition such as $-\pi < \varphi \le \pi$ or $0 \le \varphi < 2\pi$. For example, MacAnova function `atan(x,y)` returns values between $-\pi$ and $+\pi$ (–180° and +180° or –.5 and +.5 if option `angles:"degrees"` or `angles:"cycles"` is in effect). This results in discontinuities (jumps) in $\arg(X_k)$ as k varies, when X_k winds around the origin of the complex plane, even when its trajectory in the complex plane is smooth. Both `hpolar()` and `cpolar()` attempt to eliminate these jumps. They "unwind" $\arg(X_k)$ by adding or subtracting a multiple of 2π radians, 360 degrees, or 1 cycle when a jump is considered to be large. Large jumps are recognized as those for which the change in $|\arg(X_{k+1}) - \arg(X_k)|$ exceeds the value of a criterion whose default value is .75 cycles. You can specify a different value of the criterion by including keyword phrase `criterion:0.65`, for example, on `hpolar()` and `cpolar()`. The value of `criterion`

is always in cycles and must be between .5 and 1. The smoothing of jumps can be suppressed altogether by keyword phrase unwind:F.

```
Cmd> setoptions(angles:"cycles") # use cycles for angular units
Cmd> phi <- .2*run(2,6);phi# smoothly changing phi in cycles
(1)         0.4         0.6         0.8           1         1.2
Cmd> z <- cmplx(run(5)*cos(phi),run(5)*sin(phi)) # Complex series
Cmd> cpolar(z) # col 1 is modulus, column 2 is unwound phase
(1,1)           1         0.4
(2,1)           2         0.6
(3,1)           3         0.8
(4,1)           4           1
(5,1)           5         1.2
Cmd> cpolar(z,unwind:F) # suppress unwinding and you get a jump
(1,1)           1         0.4
(2,1)           2        -0.4    |(-.4)-.4| =.8 > .75
(3,1)           3        -0.2
(4,1)           4           0
(5,1)           5         0.2
```

When the columns of matrix phi are considered to be series of angles, function unwind(phi) performs the same unwinding on the columns of phi. Again you can use keyword crit to change the criterion used.

```
Cmd> @tmp <- atan(sin(phi),cos(phi)); hconcat(@tmp,unwind(@tmp))
(1,1)         0.4         0.4
(2,1)        -0.4         0.6
(3,1)        -0.2         0.8
(4,1)           0           1
(5,1)         0.2         1.2
```

5.2.5 **padto() and rotate()** We saw above that by extending a series with zero elements to length $S \geq N$, the DFT could be used to compute the CFT at equally spaced frequencies $0, 1/S, 2/S, \ldots. (S–1)/S$ cycles. To make this easier, function padto(x,S) appends sufficient rows of zeros to x to make the total number of rows S.

```
Cmd> padto(run(5),9)
(1)           1           2           3           4           5
(6)           0           0           0           0
```

Because of the implicit periodicity involved in the DFT, it is useful to be able to "rotate" a Real series so as to shift the elements in each column up or down, with any elements "pushed" off one end being moved to the other end. For any integer k, rotate(x,k) moves moves the i^{th} row of x to row i' where $i' = i + k - jN$ where the j is the unique integer such that $0 \leq i' < N$.

```
Cmd> b <- vector(3,2,1,1,2); rotate(b, 2)
(1)           1           2           3           2           1
Cmd> rotate(b, -2)
(1)           1           1           2           3           2
```

```
Cmd> rotate(b, 17)
(1)             1             2             3             2             1
```

Note that, since `length(b)` = 5 and 17 = 3×5 + 2, `rotate(b,2)` and `rotate(b,17)` are equivalent.

5.2.6 **Elementwise products of Complex and Hermitian series – hprdh(), hprshj(), cprdc(), cprdcj()** Again functions come in pairs. `hprdh(hx1,hx2)` computes the elementwise complex product of Hermitian series `hx1` and `hx2`, and `cprdc(cx1,cx2)` computes the product of Complex series `cx1` and `cx2`. `hprdhj(hx1,hx2)` and `cprdcj(cx1,cx2)` are equivalent to `hprdh(hx1,hconj(hx2))` and `cprdc(cx1,cconj(cx2))`, respectively, that is they compute elementwise products of the first argument with the complex conjugate of the second. The two arguments must be the same size and shape except that one can represent a single Hermitian or Complex series, to multiply all the series in the other argument. You can omit the second argument, in which case it is assumed to be the same as the first. Thus, `hprdhj(hx)` and `cprdcj(cx)` are equivalent to `hprdhj(hx,hx)` and `cprdcj(cx,cx)`, computing the squared amplitude of their arguments.

```
Cmd> hprdhj(hx) #same as hprdhj(hx,hx); note 0 imaginary parts
(1,1)           1             4
(2,1)          90           116
(3,1)          74           100
(4,1)           0             0
(5,1)           0             0

Cmd> cprdcj(cx) # or cprdcj(cx,cx); note 0 imaginary parts
(1,1)           5             0            25             0
(2,1)          61             0           113             0
(3,1)         181             0           265             0
(4,1)         365             0           481             0
(5,1)         613             0           761             0
```

If either of the arguments of any of these functions represents a single complex series, that series multiplies all the series in the other argument. For example, if `hx1` has 3 columns and `hx2` has 1 column, `hprdhj(hx1,hx2)` is equivalent to `hprdhj(hx1, hconcat(hx2,hx2,hx2))`. And if `cx1` has 2 columns representing a single complex series, and `cx2` has 6 columns, representing 3 complex series, then `cprdc(cx1,cx2)` is equivalent to `cprdc(hconcat(cx1,cx1,cx1),cx2)`.

5.2.7 **Discrete Fourier Transforms – rft(), hft() and cft()** `rft(rx)`, `hft(hx)`, and `cft(cx)` compute the DFT of Real, Hermitian and Complex series in the columns of `rx`, `hx`, and `cx`, respectively. Each uses the so called Fast Fourier Transform or FFT, an algorithm that allows the rapid computation of a DFT provided all the factors of N, the number of rows of the argument, are small. The particular algorithm used by MacAnova requires that no prime factors of N exceed 29 and is based on code written by Gordon Sande. In most time series applications, if the underlying data has length that does not satisfy this restriction, it can be augmented by zeros to a length S that does. `padto()` (sec. 5.2.5) allows you to do this easily. Macro `factors` available in file MacAnova.mac can be useful in finding an appropriate length.

The inversion formula involves division by N, the length of the series. To make this easier, `rft()`, `hft()`, and `cft()` all recognize keyword phrase `divbyt:T` as signalling that the result should be divided by the length. Thus, bearing in mind that the DFT of a real series is Hermitian and vice versa, the following MacAnova commands produce results equal to their arguments, `rx`, `hx` or `cx`, except for rounding error:

```
hft(hconj(rft(rx)),divbyt:T)

hconj(rft(hft(hx),divbyt:T))

cconj(cft(cconj(cft(cx)),divbyt:T))
```

Examples:

```
Cmd> rx <- run(5)^2; hx <- rft(rx)

Cmd> hx # Hermitian form of the DFT of {1, 4, 9, 16, 25}
(1)          55      -10.264      -14.736       5.6861       24.087

Cmd> cx <- cmplx(rx) ; cft(cx) # Complex form of the same
(1,1)         55            0
(2,1)    -10.264       24.087
(3,1)    -14.736       5.6861
(4,1)    -14.736      -5.6861
(5,1)    -10.264      -24.087

Cmd> hconj(rft(hft(hx),divbyt:T)) # inversion applied to hft(hx)
(1)          55      -10.264      -14.736       5.6861       24.087

Cmd> hft(hconj(rft(rx)),divbyt:T) # inversion applied to rft(hx)
(1)           1            4            9           16           25

Cmd> cconj(cft(cconj(cft(cx)),divbyt:T))#inversion applied to cft(cx)
(1,1)         1            0
(2,1)         4            0
(3,1)         9            0
(4,1)        16            0
(5,1)        25            0

Cmd> rft(run(31)) # can't do this one
ERROR: largest prime factor > 29 in length of rft
```

Suppose $N = 365 = 5\times73$ and you want to use at least $2N = 730 = 2\times5\times73$ frequencies. Since $73 > 29$ you need to find some small value of k such that $2N + k$ has all small factors. Macro `factors` can help.

```
Cmd> getmacros(factors,quiet:T) # retrieve it from MacAnova.mac

Cmd> factors(2*365+run(5)) # get factors of 731, 732, 733, 734, 735
component: composite
(1)          17          43
component: composite
(1)           2           2           3          61
component: prime
(1)         733
component: composite
(1)           2         367
component: composite
(1)           3           5           7           7
```

We see we can use 730 + 5 = 735 = 3×5×7×7.

5.2.8 **Convolving series using the DFT and function convolve()** A fundamental operation in the frequency analysis of time series is the convolution of two series. If $\{a_t\}$ and $\{X_t\}$ are finite series of length N, the *circular convolution* {Y(t)} is defined by

$$Y_t = \sum_{s=0}^{t} a_s X_{t-s} + \sum_{s=t+1}^{N-1} a_s X_{t-s+N} ,$$

where the second summation is omitted when $t = N-1$. If $\{ a_s^{(N)} \}$ and $\{ X_t^{(N)} \}$ are periodic extensions with period N, this can be expressed as

$$Y_t = Y_t^{(N)} = \sum_{s=0}^{N-1} a_s^{(N)} X_{t-s}^{(N)}$$

which also defines $\{ Y_t^{(N)} \}$, the periodic extension of $\{Y_t\}$.

If X is a m by n REAL MacAnova matrix whose columns are considered as Real series, and a is a REAL vector of length m representing a Real series, convolve(a,X) computes the circular convolution of a with each of the columns of X. It is legal for vector a to have length $r < m$, in which case it is implicitly padded with $m - r$ zeros. In fact, the principal use of convolve() is precisely the situation in which a is much shorter than X.

```
Cmd> x <- vector(1,3,2,4,5); a <- vector(1,-3,2); convolve(a,x)
(1)          -6          10          -5           4          -3

Cmd> vector(1*1+(-3)*5+2*4, 1*3+(-3)*1+2*5, 1*2+(-3)*3+2*1,\
1*4+(-3)*2+2*3, 1*5+(-3)*4+2*2) # explicit expressions
(1)          -6          10          -5           4          -3
```

There is a close relationship between circular convolution and the DFT. If $\{\hat{a}_k\}$ and $\{\hat{X}_k\}$ are the discrete Fourier transforms of two series $\{a_t\}$ and $\{X_t\}$ of the same length N, then the product series $\{\hat{a}_k \hat{X}_k\}$ is the DFT of the series which is the circular convolution of $\{a_t\}$ and $\{X_t\}$. One consequence is that circular convolution can be computed by applying the inversion formula to $\{\hat{a}_k \hat{X}_k\}$. Because of the speed of the FFT, the DFT can often compute the circular convolution of two long series faster than can convolve(). For two series of length 5000, on one computer, using convolve() took about 60 times as long as when the DFT was used.

```
Cmd> aft <- rft(padto(a,5)); xft <- rft(x)#compute DFT's of length 5

Cmd> hft(hconj(hprdh(aft,xft)),divbyt:T) #inv. transform of product
(1)          -6          10          -5           4          -3
```

If $\{a_t\}$ and $\{X_t\}$ are finite series, not necessarily the same length, their *non-circular convolution* is the series $\{Y_t\}$, where $Y_t = \sum_s a_s X_{t-s}$, the sum extending over all s such that both a_s and X_{t-s} are defined. In particular if a_t is defined for $0 \le t < r$ and X_t is

defined for $0 \le t < N$, then $Y_t = \begin{cases} 0 & t < 0 \\ \sum_{s=0}^{t} a_s X_{t-s} & 0 \le t < r-1 \\ \sum_{s=0}^{r-1} a_s X_{t-s} & r-1 \le t < N \\ \sum_{s=t-N+1}^{r-1} a_s X_{t-s} & N \le t \le N+r-1 \\ 0 & t > N+r \end{cases}$.

If $\{X_t^{(\infty)}\}$ is defined so that $X_t^{(\infty)} = X_t$ when $0 \le t < N$, and $X_t^{(\infty)} = 0$ when $t < 0$ or $t \ge N$, then

$$Y_t = \sum_{s=0}^{r-1} a_s X_{t-s}^{(\infty)},\ 0 \le t < N-1.$$

This suggests a way to compute a non-circular convolution using a circular convolution by padding the longer of the two series with enough zeros so that its length is at least equal to the combined length less 1:

```
Cmd> convolve(a,padto(x,7))
(1)          1          0         -5          4         -3
(6)         -7         10
Cmd> vector(1*1, 1*3+(-3)*1, 1*2+(-3)*3+2*1, 1*4+(-3)*2+2*3,\
1*5+(-3)*4+2*2, (-3)*5+2*4,  2*5) # explicit expressions
(1)          1          0         -5          4         -3
(6)         -7         10
```

A variant on circular convolution are the sums of *circularly lagged products* of two series of equal length N:

$$Y_t = \sum_{s=0}^{t-1} a_s X_{s-t+N} + \sum_{s=t}^{N-1} a_s X_{s-t},\ t = 0, 1, ..., N-1$$

where the first summation is omitted when $t = 0$. This is almost the same as a circular convolution with $\{X_t\}$ in reverse order. Sums of circularly lagged products can be computed by `convolve()` using keyword phrase `reverse:T`. As before, if the first argument is shorter than the second it is implicitly extended with zeros.

```
Cmd> convolve(a,x,reverse:T)
(1)         -4          8         -9          0          5
Cmd> vector(1*1+(-3)*3+2*2, 1*5+(-3)*1+2*3, 1*4+(-3)*5+2*1,\
1*2+(-3)*4+2*5, 1*3+(-3)*2+2*4) # explicit expressions
(1)         -4          8         -9          0          5
```

The DFT of a series of circularly lagged sums is the product of the DFT of the first factor times the complex conjugate of the DFT of the second, that is, $\hat{Y}_k = \hat{a}_k \overline{\hat{X}_k}$. Consequently, the inversion formula can be used to compute sums of circularly lagged products from DFT's.

```
Cmd> hft(hconj(hprdhj(aft,xft)),divbyt:T)
(1)         -4          8         -9          0          5
```

The non-zero elements of the series of *non-circularly* lagged products

$$Z_t = \sum_s a_s X^{(\infty)}_{s-t}\,,\ t = 0, \pm1, \pm2, \ldots,$$

where $\{X^{(\infty)}_t\}$ is as above, can be computed from circularly lagged products by padding the second factor with enough zeros to make its length S greater than the sum of the lengths minus 1:

```
Cmd> convolve(a,padto(x,7),reverse:T)
(1)          -4           3           2           5         -11
(6)           0           5

Cmd> vector(1*1+(-3)*3+2*2, (-3)*1+2*3, 2*1, 1*5, 1*4+(-3)*5,\
1*2+(-3)*4+2*5, 1*3+(-3)*2+2*4) # explicit expressions
(1)          -4           3           2           5         -11
(6)           0           5
```

These values are in the order Z_0, Z_1, Z_2, Z_{-4}, Z_{-3}, Z_{-2}, Z_{-1}. Alternatively, using the DFT,

```
Cmd> hft(hconj(hprdhj(rft(padto(a,7)),rft(padto(x,7)))),divbyt:T)
(1)          -4           3           2           5         -11
(6)  7.613e-16           5
```

5.3 **Functions related to time domain time series analysis** At present, MacAnova has relatively few commands directly related to the time domain analysis of time series. `movavg()` and `autoreg()` implement moving average (MA) and autoregressive (AR) operators in the Box-Jenkins sense. They can be used to generate artificial univariate ARIMA data, to compute innovations from a ARIMA model, given its coefficients, and to compute the spectrum of an ARMA process. They also have some purely mathematical uses, such as computing the coefficients of power series representing the reciprocal of a polynomial or the ratio of two polynomials, computing successive differences, and computing cumulative sums. `yulewalker()` and `partacf()` allow computing autoregression coefficients and partial autocorrelation functions from autocorrelation functions and vice versa. `toeplitz()` transforms an autocovariance function into a variance/covariance matrix.

5.3.1 **Moving average and autoregressive operators** A vector $[\theta_1, \theta_2, \ldots, \theta_q]'$ of real constants defines a *moving average* (MA) operator of order q acting on a series $\{X_t\}$, transforming it to the series $\{Y_t\}$, where $Y_t = X_t - \sum_{s=1}^{q} \theta_s X_{t-s}$. Let $\theta(z)$ be the polynomial $\theta(z) \equiv 1 - \sum_{s=1}^{q} \theta_s z^s$. Then the MA operator can also be expressed as $\{Y_t\} = \theta(B)\{X_t\}$, where B is the backward lag operator, defined by $BX_t = X_{t-1}$, $B^2X_t = X_{t-2}$, ... If $q = 1$ and $\theta_1 = 1$, then $\theta(B) = 1 - B$ and the MA operator is the backwards *first difference operator* transforming X_t to $X_t - X_{t-1}$.

If $\{c_t\}$ is a sequence with $c_t = 0$ for $t < 0$, define $f(z) = \sum_{t=0}^{\infty} c_t z^t$ to be a formal power series in z. Then the sequence $\{d_t\} = \theta(B)\{c_t\}$ consists of the coefficients of the function $\theta(z)f(z)$ considered as a power series in z. In particular, if $c_t = 0$ for $t > m$, so that $f(z)$ is a polynomial of degree at most m, then applying the MA operator to $\{c_t\}$ corresponds to computing the coefficients of the product of two polynomials.

If $\theta_1(B)$ and $\theta_2(B)$ are polynomials of degree q_1 and q_2 corresponding to two MA operators of length q_1 and q_2, respectively, then the polynomial $\theta_1(B)\theta_2(B)$ of degree $q_1 + q_2$ corresponds to the operator obtained by first transforming a series using $\theta_1(B)$ and then transforming the output using $\theta_2(B)$.

A vector $[\varphi_1, \varphi_2, ..., \varphi_p]'$ of real constants defines an *autoregressive* (AR) operator of order p, transforming a series $\{X_t\}$ to the series $\{Y_t\}$, defined recursively by $Y_t = X_t + \sum_{s=1}^{p} \varphi_s Y_{t-s}$, or, equivalently, by $\varphi(B)Y_t \equiv Y_t - \sum_{s=1}^{p} \varphi_s Y_{t-s} = X_t$, where $\varphi(z)$ is the polynomial $\varphi(z) = 1 - \sum_{s=1}^{p} \varphi_s z^s$. This is conveniently expressed symbolically as $Y_t = \varphi(B)^{-1}X_t$, where $\varphi(B)^{-1}$ represents the operator inverse to $\varphi(B)$, that is $\varphi(B)^{-1}\varphi(B) = 1$. $\varphi(B)^{-1}$ can also be expressed as $\varphi(B)^{-1} = 1 + \sum_{s=1}^{\infty} \psi_s B^s$ where $1 + \sum_{s=1}^{\infty} \psi_s z^s$ is the formal power series expansion such that $\left(1 + \sum_{s=1}^{\infty} \psi_s B^s\right) \times \left(1 - \sum_{s=1}^{p} \varphi_s z^s\right) = 1$.

Because of the recursive nature of this definition, you generally need to consider an AR operator as acting only on values of X_t for $t \geq t_0$, with specified "starting values" Y_{t_0}, Y_{t_0-1}, Y_{t_0-2}, ..., Y_{t_0-p}. If $p = 1$ and $\varphi_1 = 1$, then $Y_t = Y_{t-1} + X_t$ and the AR operator corresponds to the *cumulative sum*

$$Y_t = Y_{t_0} + X_{t_0+1} + X_{t_0+2} + \quad X_t = Y_{t_0} + \sum_{s=t_0+1}^{t} X_s \text{ for } t > t_0.$$

As before, let $\{c_t\}$ be a sequence with $c_t = 0$ for $t < 0$ and let $f(z) = \sum_{t=0}^{\infty} c_t z^t$. Then specifying starting values $c_{-1} = c_{-2} = ... = c_{-p} = 0$, the sequence $\varphi(B)^{-1}\{c_t\}$ consists of the coefficients in the power series representation for $f(z)/\varphi(z)$. If $f(z)$ is a polynomial, then $\{c_t\}$ are the power series coefficients of a rational function. If $X_0 = 1$ and $X_t = 0$, $t \neq 0$, then $\{c_t\}$ is the sequence of the power series coefficients of $1/\varphi(z)$.

A time series $\{X_t\}$ with mean μ_t satisfying $\varphi(B)(X_t - \mu_t) = a_t$, where $\{a_t\}$ is a sequence of independent identically distributed random variables with mean 0 is called an AR(p) (*autoregressive* series of order p) series. When $\{X_t\}$ satisfies $X_t = \mu_t + \theta(B)a_t$, it is called a MA($q$) (*moving average* series of order q) series. When $\{X_t\}$ satisfies $\varphi(B)(X_t - \mu_t) = \theta(B)a_t$ it is called an ARMA(p,q) (*autoregressive-moving average*) series. See Box and Jenkins (1976).

5.3.2 **movavg()** If theta is a REAL vector of length q and a is a matrix, then movavg(theta,a) applies the order q MA operator specified by theta to each column of a, with a[k,] assumed 0 for k < 1. That is, the output is a matrix x of the same size and shape as a with

$$x[i,j] = a[i,j] - \sum_{k=1}^{q} (\text{theta}[k]*a[i-k,j]), \text{ with } a[l,j] = 0 \text{ for } l < 1$$

```
Cmd> setseeds(287236458,760033449) # set seeds (See Sec. 2.12)
Cmd> theta <- vector(.5, .3); n <- 5; a <- rnorm(n); a
(1)       1.212      0.63568      -1.8396      -0.5317      0.41061
Cmd> movavg(theta,a)
(1)       1.212     0.029677      -2.5211       0.1974       1.2283
```

An equivalent, but much clumsier, computation is

```
Cmd> a-theta[1]*vector(0,a[-n])-theta[2]*vector(0,0,a[-run(n-1,n)])
(1)       1.212     0.029677      -2.5211       0.1974       1.2283
```

You also can use movavg() to compute first and higher order differences of a series.

```
Cmd> d1 <- movavg(1,a); d1 # first differences a[i]-a[i-1] of a
(1)       1.212     -0.57633      -2.4753       1.3079      0.94231
Cmd> d2 <- movavg(vector(2,-1),a); d2 # second differences of a
(1)       1.212      -1.7883       -1.899       3.7832      -0.3656
```

Note that d1[1] = a[1] - 1*0 = a[1], d2[1] = a[1] - 2*0 + 1*0 = a[1], and d2[2] = a[2] - 2*a[1] + 1*0 = a[2] - 2*a[1]. The second difference computation works because $(a_t - a_{t-1}) - (a_{t-1} - a_{t-2}) = a_t - 2a_{t-1} - (-1)\, a_{t-2}$.

Sometimes you may need to apply a moving average operator *backwards*, that is to carry out the computation

$$x[i,j] = a[i,j] - \sum_{k=1}^{q} (\text{theta}[k]*a[i+k,j]), \text{ with } a[l,j]=0 \text{ for } l > \text{nrows}(a)$$

You do this using keyword phrase reverse:T.

```
Cmd> movavg(theta,a,reverse:T)
(1)       1.4461        1.715      -1.6969     -0.73701      0.41061
Cmd> a - theta[1]*vector(a[-1],0) - theta[2]*vector(a[-run(2)],0,0)
(1)       1.4461        1.715      -1.6969     -0.73701      0.41061
```

In some situations you may need to restrict a moving average operator to a subset of the rows of the input matrix. If b is a matrix with the same size and shape as a and i1 and i2 are integers between 1 and nrows(a), the output of

```
movavg(theta,a,limits:vector(i1,i2),start:b)
```

is computed as just described only for rows i1, i1+1,..., i2. The remaining rows are copied from rows 1 to i1-1 and rows i2+1 to nrows(x) of matrix b.

```
Cmd> movavg(theta,a,limits:vector(2,4),start:run(5))
(1)          1     0.029677      -2.5211       0.1974            5
```

```
Cmd> movavg(theta,a,limits:vector(2,4),start:run(5),reverse:T)
(1)           1        1.715      -1.6969     -0.73701           5
```

In both cases, rows i1 through i2 of b are ignored.

Function movavg() is the inverse of autoreg() (Sec. 5.3.3) and vice versa, in that, if movavg() is applied to the output x of autoreg() with the same coefficients, direction, and starting values, if any, the input a is recovered, except for rounding error.

5.3.3 **autoreg()** If phi is a REAL vector of length p and a is a matrix, autoreg(phi,a) applies the order p AR operator specified by vector phi to each column of a, with any needed starting values assumed 0. The output is a matrix x of the same size and shape as a with

$$x[i,j] = a[i,j] + \sum_{k=1}^{p} (phi[k]*x[i-k,j]), \text{ with } x[l,j] = 0 \text{ for } l < 1.$$

```
Cmd> phi <- vector(1.43,-.57) ; x <- autoreg(phi,a); x
(1)       1.212       2.3689        0.857     -0.65643      -1.0166
Cmd> a + phi[1] * vector(0,x[-5]) + phi[2]*vector(0,0,x[-run(4,5)])
(1)       1.212       2.3689        0.857     -0.65643      -1.0166
Cmd> autoreg(1,a) # cumulative sums a[1], a[1]+a[2], ...
(1)       1.212       1.8477    0.0080789     -0.52362     -0.11301
```

It is often important to be able to specify starting values for an AR transformation. If b is a matrix which is the same size and shape as a and i1 and i2 are integers between 1 and nrows(a), the output of

```
autoreg(phi,a,limits:vector(i1,i2),start:b)
```

is computed as just described only for rows i1, i1+1,..., i2, with rows 1, ..., i1-1 of b copied to the output and then used as starting values, and with rows i2+1,...,nrows(a) of b copied to the corresponding rows of the output but otherwise ignored.

```
Cmd> b <- run(5);x1 <-  autoreg(phi,a,limits:vector(2,4),start:b);x1
(1)           1       2.0657      0.54431     -0.93077           5
Cmd> a[3]+phi[1]*x1[2]+phi[2]*x1[1] # x1[1] is the same as b[1]
(1)     0.54431   Same as x1[3]
```

autoreg(phi,a,reverse:T) applies the autoregressive operator in reverse, that is

$$x[i,j] = a[i,j] + \sum_{k=1}^{p} (phi[k]x[i+k,j]), \text{ with } x[l,j] = 0 \text{ for } l > nrows(a).$$

```
Cmd> x1r <- autoreg(phi,a,limits:vector(2,4),start:b,reverse:T);x1r
(1)           1       3.6909       4.7746       6.6183           5
Cmd> a[3]+phi[1]*x1r[4]+phi[2]*x1r[5] # x1r[5] is the same as b[5]
(1)      4.7746   Same as x1r[3]
```

In both cases, rows i1 through i2 of b are ignored.

One way to describe the action of `autoreg()` is to note that it computes a solution to the inhomogeneous linear difference equation $X_t - \sum_{s=1}^{p} \varphi_s X_{t-s} = a_t$, with starting values $X_t = 0$ for $t < 0$. The Fibonacci numbers $\{F_t\}$ with $F_0 = F_1 = 1$ and $F_t = F_{t-2} + F_{t-2}$ satisfy such an equation with $\varphi_1 = 1$ and $\varphi_2 = 1$ and $a_0 = 1$, $a_t = 0$, $t \neq 0$. Thus the first 10 Fibonacci numbers are computed by:

```
Cmd> autoreg(vector(1,1),padto(1,10)) # compute 10 Fibonacci numbers
 (1)          1           1           2           3           5
 (6)          8          13          21          34          55
```

These are also the starting coefficients of the power series for $1/(1-z-z^2) = 1 + z + 2z^2 + 3z^3 + 5z^4 + \ldots$.

`autoreg()` is the inverse of `movavg()` and vice versa, in that, when `autoreg()` is applied to the output `x` of `movavg()` with the same coefficients, direction, and starting values, if any, the input `a` is recovered, except for rounding error.

If `a` is a vector of independent random variables with zero mean, perhaps produced by `rnorm()` (See. Sec. 2.13), you can use `movavg()` and `autoreg()` to generate an ARMA(p,q) time series . If `phi` and `theta` are vectors that contain the autoregressive and moving average coefficients, respectively, then `y <- autoreg(phi, movavg(theta,a))` or `y <- movavg(theta,autoreg(phi,a))` computes an ARMA series with innovation series {..., 0, 0,..., `a[1]`, `a[2]`, `a[3]`, ..., `a[nrows(a)]`}. To avoid the effect of transients from the zero starting values, it may be wise to discard some values from the start of `y`.

Conversely, using the fact that `autoreg()` and `movavg()` are inverses of one another, the innovation series can be recovered from `y` by `movavg(phi,autoreg(theta,y))` or `autoreg(theta,movavg(phi,a))`.

5.3.4 **yulewalker() and partacf()** If $\{X_t\}$ is a stationary time series with expectation μ and autocorrelations $\rho_s = \text{corr}[X_t, X_{t+s}] = \rho_{-s}$, and you want to predict X_t by an expression of the form $\mu + \sum_{k=1}^{p} \varphi_{kp}(X_{t-k} - \mu)$ depending linearly on the p preceding observations X_{t-1}, X_{t-2}, ..., X_{t-p}, then the optimal values (in the sense of minimum variance of the prediction error) for φ_{1p}, φ_{2p}, ..., φ_{pp} can be shown to satisfy the Yule-Walker equations

$$\rho_s = \sum_{k=1}^{p} \varphi_{kp}\rho_{s-k}\ ,\ s = 1, 2, \ldots, p$$

The sequence φ_{11}, φ_{22}, φ_{33}, ... is the *partial autocorrelation function* of $\{X_t\}$. It can further be shown that the prediction variance $\sigma_p^2 = V[X_t - \mu - \sum_{k=1}^{p} \varphi_{kp}(X_{t-k} - \mu)]$ satisfies

$$\sigma_p^2 = \prod_{k=1}^{p} (1 - \varphi_{kk}^2) V[X_t]$$

There are fairly stringent conditions that a sequence $\{\rho_1, \rho_2, \ldots\}$ of autocorrelations must satisfy. For example $|\rho_s| \le 1$, $1 - 2\rho_1^2 + \rho_2 > 0$, $1 - \rho_1^2 - \rho_2^2 - \rho_3^2 + 2\rho_1\rho_2\rho_3 > 0$, ... The conditions may also be expressed as $|\varphi_{kk}| \le 1$, $k = 1, 2, \ldots$, where $\varphi_{11}, \varphi_{22}, \ldots, \varphi_{pp}$ are the partial autocorrelations. Moreover, if $|\varphi_{kk}| = 1$, $\varphi_{jj} = 0$, for $j > k$.

In particular, if $\{X_t\}$ is an AR(p) series with autoregression coefficients $\{\varphi_1, \ldots, \varphi_p\}$ and innovation variance σ^2, $\varphi_{pp} = \varphi_p$, $\varphi_{kk} = 0$ for $k > p$, $\sigma_p^2 = \sigma_{p+1}^2 = \sigma_{p+2}^2 = \ldots = \sigma^2$ and the best linear prediction of X_t is $\mu + \sum_{k=1}^{p} \varphi_k (X_{t-k} - \mu)$. Thus, the coefficients $\{\varphi_1, \ldots, \varphi_p\}$can be recovered from $\rho_1, \ldots, \rho_p$ by solving $\rho_s = \sum_{k=1}^{p} \varphi_k \rho_{s-k}$, $s = 1, 2, \ldots, p$.

`yulewalker(rho)`, where `rho` is a `REAL` matrix, solves the Yule-Walker equations for each column of `rho`, considered as the first `nrows(rho)` autocorrelations of a stationary time series. If a column of `rho` violates the conditions for it to be a sequence of autocorrelations, then `yulewalker()` does the best it can. If k is the first index such that $|\varphi_{kk}| > 1$, then the corresponding column of the result is set to $\varphi_{1k}, \varphi_{2k}, \ldots, \varphi_{k-1,k}$, $\varphi_{kk} / |\varphi_{kk}|$, 0, 0, ... and a warning message is printed.

`yulewalker(phi,inverse:T)` is the inverse operation. If a column of `phi` contains the coefficients of a stationary autoregressive series, then the corresponding column of the result contains the autocorrelations. Again there are fairly stringent conditions for this to be possible; if these are not satisfied for a column of `phi`, the corresponding column of the result is set to 0 and a warning message is printed.

```
Cmd> rho <- vector(0.2,-0.248,0.31232)
Cmd> phi <-yulewalker(rho); phi
(1)         0.41        -0.43          0.5
Cmd> yulewalker(phi,inverse:T) # recover rho from phi.
(1)          0.2       -0.248      0.31232
Cmd> yulewalker(vector(.1,-.7,.29,.4))
WARNING: Col. 1 of argument not positive definite at row 3
         Remaining elements in column assumed zero
(1)      0.88889     -0.88889            1            0
Cmd> movavg(phi,vector(rho[run(3,1)],1,rho))[run(5,7)]
(1)  -1.3878e-17  -2.7756e-17            0
```

The last line shows that `phi` satisfies the Yule-Walker equations to within rounding error.

5.3.5 **toeplitz()** A $m \times m$ Toeplitz matrix $A = [a_{ij}]$ is one with constant elements down each diagonal, that is $a_{ij} = d_{i-j}$. For a symmetric Toeplitz matrix, $d_{-k} = d_{k-1} = a_{k1}$, $k = 1, \ldots, m$. If $\{X_t\}$ is a covariance stationary time series with autocovariances $\{\gamma_s\}$, $\gamma_s = \mathrm{Cov}[X_t, X_{t+s}] = \gamma_{-s}$, then the covariance matrix of any m consecutive values $\{X_t\}_{t=t_0}^{t_0+m-1}$ is a $m \times m$ symmetric Toeplitz matrix with γ_0 down the main diagonal. Similarly, if $\rho_s = \gamma_s / \gamma_0$, $= 0, \pm 1, \pm 2, \ldots$ are autocorrelations, the correlation matrix of $\{X_t\}_{t=t_0}^{t_0+m-1}$ is a $m \times m$ symmetric Toeplitz matrix.

`toeplitz(d)` returns a symmetric Toeplitz matrix whose first column is the same as REAL vector `d`. The principal use of `toeplitz(d)` is to create a covariance or correlation matrix from the autocovariance or autocorrelations of a time series.

```
Cmd> toeplitz(vector(1,.6,-.17,-.52,-.2))
(1,1)          1          0.6        -0.17        -0.52         -0.2
(2,1)        0.6            1          0.6        -0.17        -0.52
(3,1)      -0.17          0.6            1          0.6        -0.17
(4,1)      -0.52        -0.17          0.6            1          0.6
(5,1)       -0.2        -0.52        -0.17          0.6            1
```

5.3.6 **Finding zeros of polynomials – polyroot()** Some important properties of moving average and autoregressive operators depend on the values of the zeros of their associated polynomials. Let $\theta(z) = 1 - \sum_{s=1}^{q} \theta_s z^s$ be the polynomial associated with a MA operator and let $\{a_t\}$ be a "white noise" series, that is a sequence of independent identically distributed random variables. Then the MA(q) series $\{Y_t\}$ defined by $Y_t = a_t - \sum_{s=1}^{q} \theta_s a_{t-s}$ is "invertible" if and only if all the zeros of $\theta(z)$ are outside the unit circle, that is $|z| > 1$ for all z such that $\theta(z) = 0$. Similarly, if $\varphi(z) = 1 - \sum_{s=1}^{p} \varphi_s z^s$ is the polynomial associated with an AR operator and the AR(p) series $\{Y_t\}$ is defined by $Y_t = a_t + \sum_{s=1}^{p} \varphi_s Y_{t-s}$, then $\{Y_t\}$ can be stationary only when all the zeros of $\varphi(z)$ are outside the unit circle.

`polyroot(c)` computes the real and possibly complex roots of the polynomial $P(x) = x^n - \sum_{k=1}^{n} c_k x^{n-k} = x^n Q(1/x)$, where $Q(t) = 1 - \sum_{k=1}^{n} c_k t^k$, with c_k = `c[k]`. It is clear that $Q(t)$ has all its zeros *outside* the unit circle if and only if $P(x)$ has all its zeros *inside* the unit circle. If `c` is n by m, the result returned is a n by $2m$ matrix with the real and imaginary parts of the roots associated with column j of `c` in columns $2j-1$ and $2j$, that is, in Complex form.

```
Cmd> phi <- vector(1.42, -.73)

Cmd> z <- polyroot(phi); z # zeros of z^2 - 1.42z + .73
(1,1)         0.71      0.47529
(2,1)         0.71     -0.47529

Cmd> cprdc(z) - phi[1]*z - cmplx(phi[2]) # they are zeros
(1,1)  -1.1102e-16            0
(2,1)  -1.1102e-16            0

Cmd> creal(cpolar(z)) # modulus of zeros of z^2-phi[1]*z-phi[2]
(1)       0.8544       0.8544
```

Since the modulus of the zeros of $z^2 - \varphi_1 z - \varphi_2$ are inside the unit circle, the roots of $\varphi(z) = 1 - \varphi_1 z - \varphi_2 z^2$ are outside the unit circle and $Y_t = \varphi_1 Y_{t-1} - \varphi_2 Y_{t-2} + a_t$ is stationary.

5.4 **Macros useful in time series analysis** Several macros that are useful in time series analysis are in file `Tser.mac` which is distributed with MacAnova. Note that these must be read in using `macroread()` (Sec. 7.5.1) or macro `getmacros` (Sec. 7.5.3).

Some of the macros use the value of variable `DELTAT`, when it exists, as the interval Δt between observations. When it does not exist, Δt is assumed to be 1. `DELTAT` is pre-defined to have value 1. Some macros use the value of variable `S`, when it exists, as the number S of frequencies at which frequency functions such as spectra or cross spectra will be computed. `S` is *not* a pre-defined variable.

In the examples below we use artificial data generated by the following MacAnova commands.

```
Cmd> phi <- vector(1.42, -.73)# coefs of oscillatory AR(2) series
Cmd> theta <- .7 # coef of MA(1) series
Cmd> setseeds(2013847346, 1009143553)# set seeds (Sec. 2.13.1)
Cmd> x <- 12 + 2*autoreg(phi,rnorm(250))[-run(50)] # Sec. 5.3.3
Cmd> y <- 50 + 4*movavg(theta,rnorm(201))[-1] # Sec. 5.3.2
Cmd> z <- .5*x + y
```

When the time interval $\Delta t = 1$, the spectrum of an AR(p) series with coefficients $[\varphi_1, \varphi_2, ..., \varphi_p]$ is $S(f) = \dfrac{\sigma^2}{\left|1 - \sum_{s=1}^{p} \varphi_s e^{-i2\pi sf}\right|^2}$ and the spectrum of a MA(q) series with coefficients $[\theta_1, \theta_2, ..., \theta_q]$ is $S(f) = \sigma^2 \left|1 - \sum_{s=1}^{q} \theta_s e^{-i2\pi sf}\right|^2$, where in both cases σ^2 is the innovation or error variance. Since the expressions inside $|...|$ are continuous Fourier transforms of the sequences $\{1, -\varphi_1, -\varphi_2, ..., -\varphi_p\}$ or $\{1, -\theta_1, -\theta_2, ..., -\theta_q\}$, these spectra can be computed at equally spaced frequencies in MacAnova.

```
Cmd> S <- 400
Cmd> sxx <- hreal(2^2*hprdhj(rft(autoreg(phi,padto(1,S)))))
Cmd> syy <- hreal(4^2*hprdhj(rft(movavg(theta,padto(1,S)))))
Cmd> szz <- .5^2*sxx + syy # o.k. because x and y independent
```

Because the series in `x` and `y` are independent, $S_{ZZ}(f) = (.5)^2 \times S_{XX}(f) + S_{YY}(f)$ and the cross spectrum of `x` and `z` is $S_{XZ}(f) = .5 \times S_{XX}(f)$. Hence the phase $\arg(S_{XZ}(f))$ of the cross spectrum is 0 and the coherence is $\dfrac{.5S_{XX}(f)}{\sqrt{S_{XX}(f)(.25S_{XX}(f) + S_{YY}(f))}} = 1\Big/\sqrt{1 + 4S_{YY}(f)/S_{XX}(f)}$.

```
Cmd> coher <- 1/sqrt(1+4*syy/sxx)
```

Spectra, cross-spectra and coherence are examples of *frequency functions* defined in the frequency domain measured in cycles per unit time. Frequency functions derived from

time series observed every Δt time units are generally periodic with period $1/\Delta t$. However, because they generally are symmetric about zero ($g(-f) = g(f)$), anti-symmetric about zero ($g(-f) = -g(f)$) or are complex satisfying Hermitian symmetry ($g(-f) = \overline{g(f)}$), they are completely specified by their values between 0 and $.5/\Delta t$ cycles. Frequency $.5/\Delta t$ is known as the Nyquist frequency.

5.4.1 **Plotting against time – tsplot** This macro provides an interface to functions `lineplot()`, `plot()` and `chplot()` for plotting time series and frequency functions. On both of them, you can use the usual keywords `title`, `xlab`, `ylab`, xmin, `xmax`, ymin, `ymax`, ... (see Sec. 8.5.1 and 8.5.2).

`tsplot` makes plots of the columns of its first argument against equally spaced values.

The simplest usage is simply `tsplot(y)`, where `y` is a `REAL` vector or matrix. This plots rows 1, 2, ..., `nrows(y)` against 0, Δt, $2\Delta t$, ..., `(nrows(y)-1)`$\times\Delta t$, where Δt is the value of variable `DELTAT`. If `DELTAT` does not exist, $\Delta t = 1$. `tsplot(y,t0)` does the same, except the rows are plotted against `t0`, `t0` + Δt, `t0` + $2\Delta t$, ..., `t0` + `(nrows(y)-1)`$\times\Delta t$. Finally, `tsplot(y,t0,deltat)` does the same except that Δt = `deltat`.

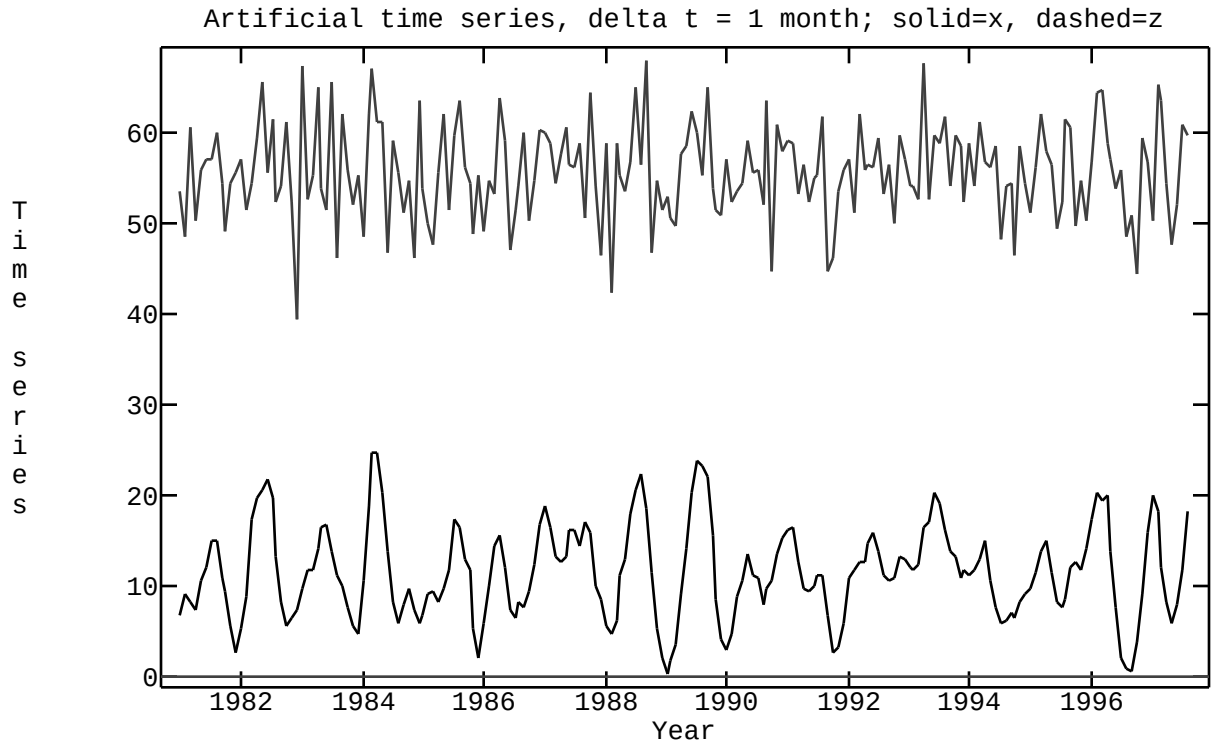
In any of these usages, you can also include one or more of the following keyword phrases

Keyword Phrase	Meaning
`lines:F`	Suppress connecting lines
`impulse:T`	Add impulses (lines connecting data point with y = 0 line)
`char:CharVar`	Use `chplot()` with `CHARACTER` variable `CharVar` as argument 3

With `lines:F` but without `char:CharVec`, `tsplot` uses `plot()`. `lines:F` and `char:T` won't work with versions of `tsplot` earlier than June 1998.

```
Cmd> getmacros(tsplot,quiet:T)
```

```
Cmd> tsplot(hconcat(x,z),1981,1/12,xlab:"Year",title:\
"Artificial time series, delta t = 1 month; solid=x, dashed=z")
```



This plots `x` and `y` as if they represented monthly data ($\Delta t = 1/12$) starting January 1981. See Sec. 5.4.3 and 5.4.5 for other examples of the use of `tsplot`.

5.4.2 **Plotting against frequency – ffplot** `ffplot(y)` or `ffplot(y,0)` make line plots of the columns of `y` against equally spaced values interpreted as *frequencies*, 0, $1/(S\times\Delta t)$, $2/(S\times\Delta t)$, ..., cycles per unit time, where Δt is the value of `DELTAT` when `DELTAT` is defined or 1 when it is not, and S = `nrows(y)`. The default frequency range is $0 \le f \le .5/\Delta t$, that is, $S/2 + 1$ or $(S+1)/2$ different frequencies, depending on whether S = is even or odd.

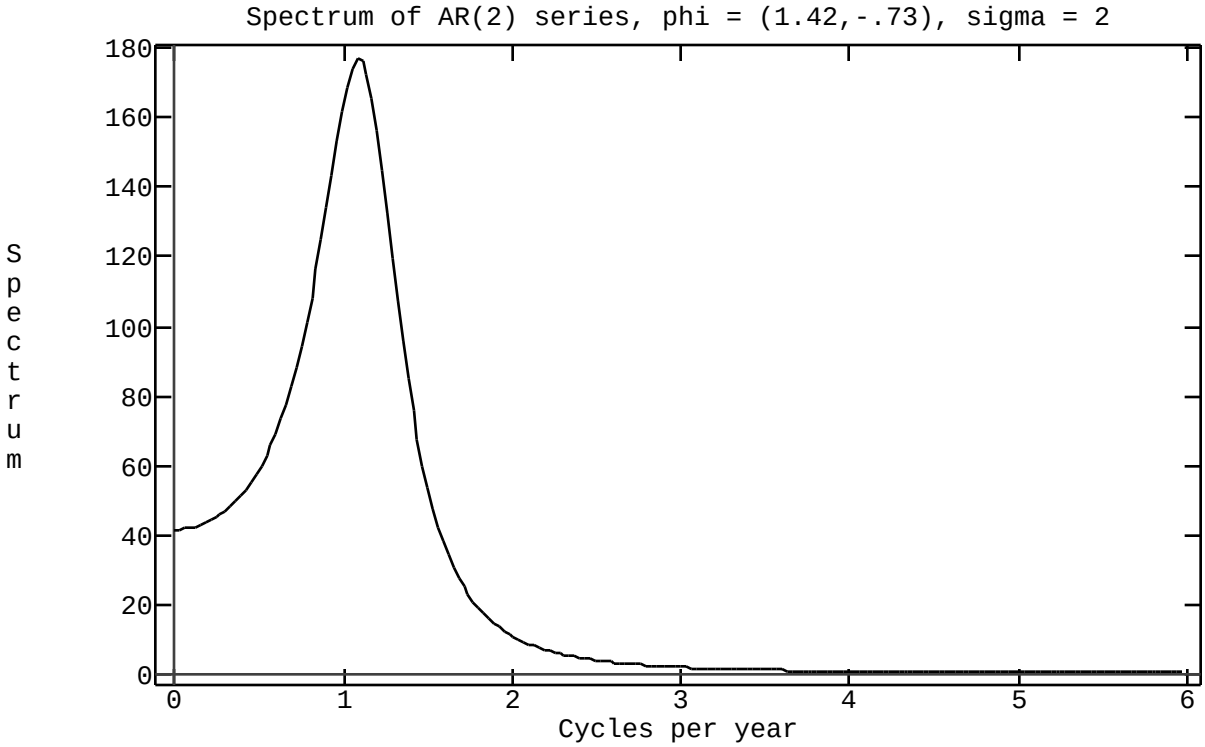
The columns of `y` are typically frequency functions such as periodograms (see Sec. 5.4.6), spectra, or coherences computed at S frequencies from one or more time series observed at intervals of Δt time units.

`ffplot(y,range)`, where `range` = `vector(f1,f2)`, does the same except the plot is between frequencies `f1` and `f2` cycles per unit time. `ffplot(y,f)`, where `f` $\neq 0$ is a `REAL` scalar, is equivalent to `ffplot(y,vector(0,f))` or `ffplot(y,vector(f,0))`, depending on the sign of `f1`. When `DELTAT` is defined, `ffplot(y,.5/DELTAT)` produces the same plot as `ffplot(y)`. Values for `f1` and `f2` need not be restricted to the interval $(0, 1/\Delta t)$. If either element of `range` is outside this interval, the columns of `y` are assumed to represent periodic frequency functions with period $1/\Delta t$ (one cycle every S rows) and their values are periodically extended.

`ffplot(y,range,deltat)` does the same, except that the value of `REAL` scalar `deltat` overrides the default value for Δt.

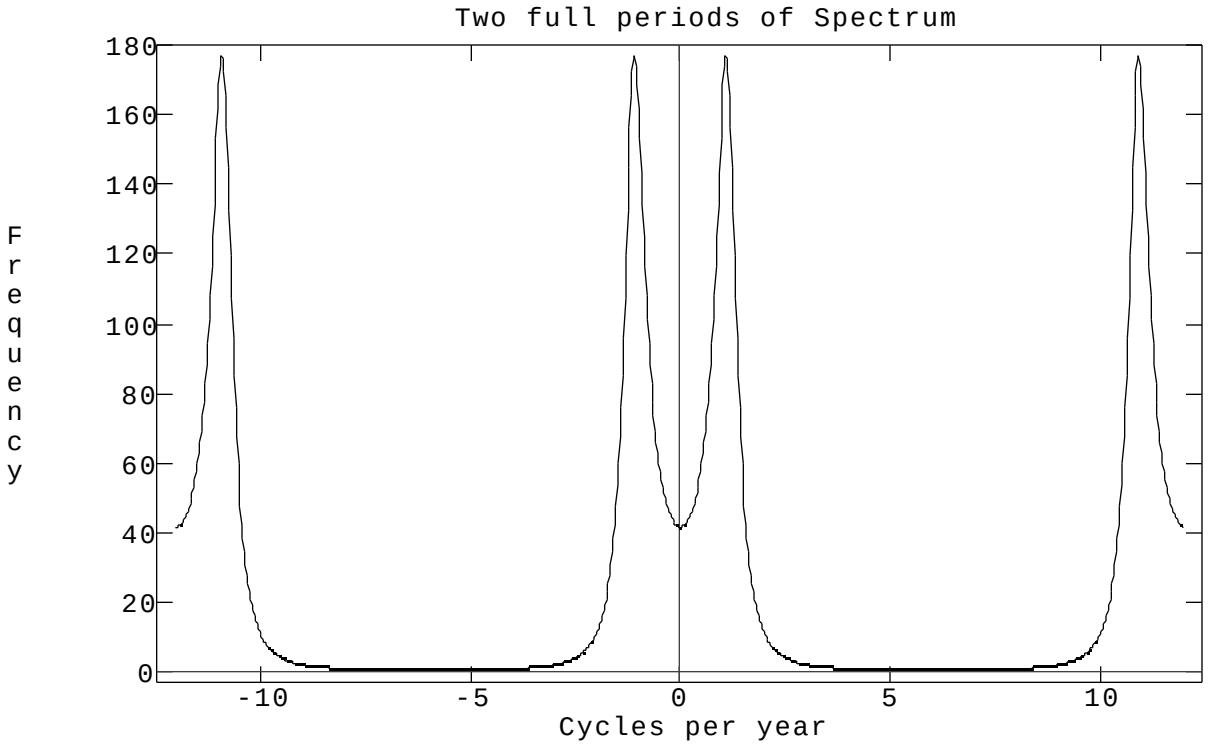
You can use keyword phrases `lines:F`, `impulse:T` and `char:T` as with `tsplot` (Sec. 5.4.1).

```
Cmd> DELTAT <- 1/12 # observations every month = 1/12 year
Cmd> getmacros(ffplot,quiet:T)
Cmd> ffplot(sxx,xlab:"Cycles per year",ylab:"Spectrum",\
title:"Spectrum of AR(2) series, phi = (1.42,-.73), sigma = 2")
```

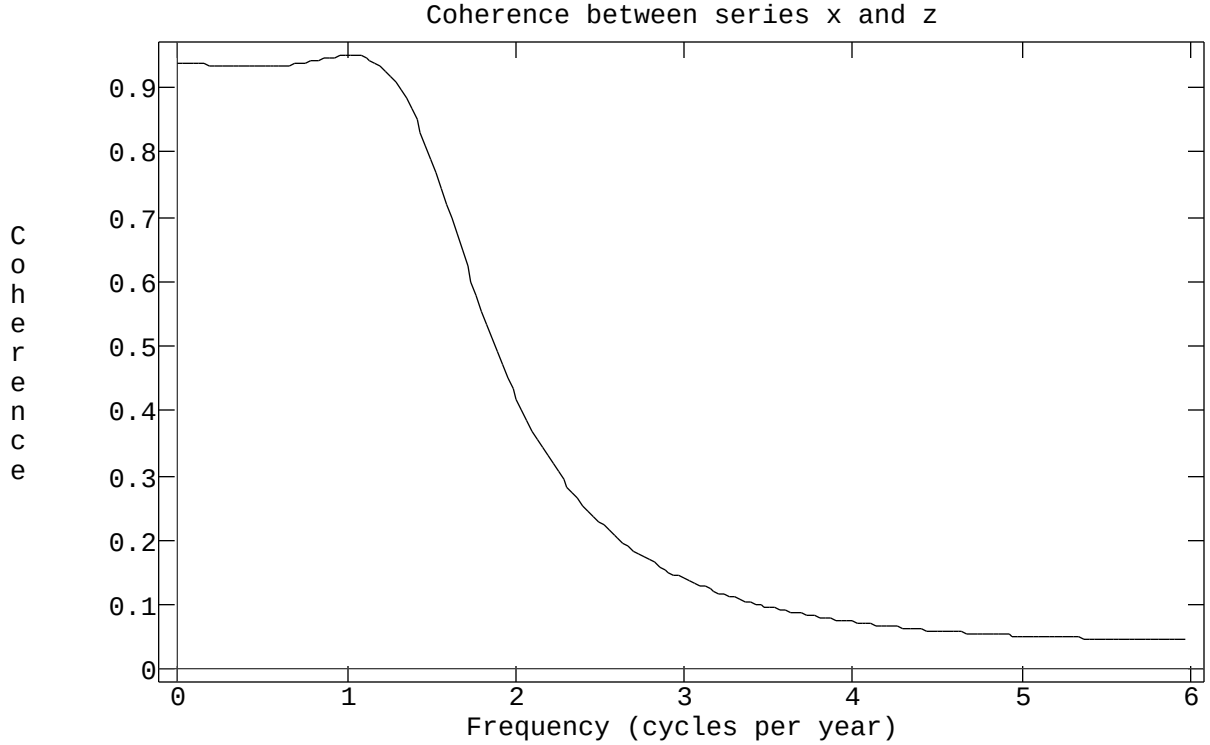


Here is an example exploiting the capability of `ffplot` to extend its first argument periodically.

```
Cmd> ffplot(hreal(sxx),vector(-1,1)/DELTAT,xlab:"Cycles per year",\
ylab:"Frequency",title:"Two full periods of Spectrum")
```




```
Cmd> ffplot(coher,ymin:0,xlab:"Frequency (cycles per year)",\
ylab:"Coherence",title:"Coherence between series x and z")
```

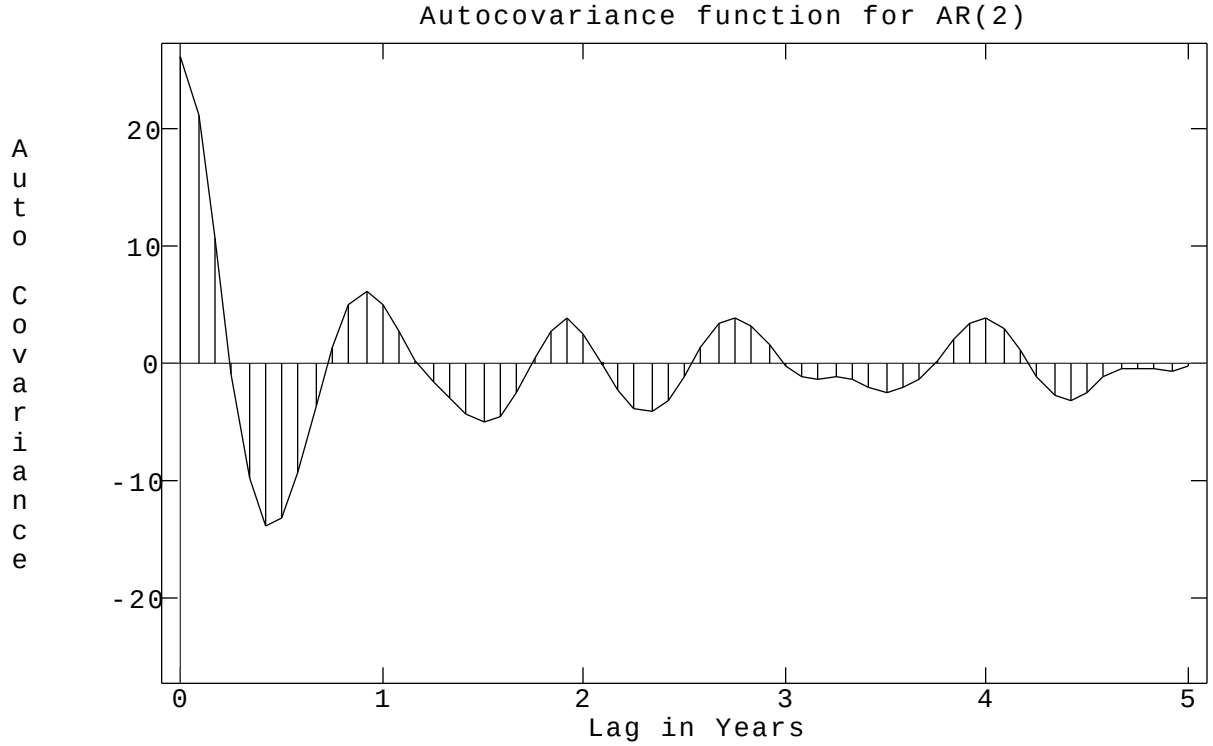


5.4.3 **Computing auto-covariances – autocov** The sample autocovariance function of a time series can be computed by macro `autocov`. Its usage is `acf <- autocov(x, nlags,L)`, where `x` is a `REAL` matrix whose columns are considered as a Real series. For each column, `autocov` computes the sample autocovariance function

$$C_s = \frac{1}{N}\sum_{t=0}^{N-s-1}(X_t - \overline{X})(X_{t+s} - \overline{X}),\ s = 0, 1, ..., \texttt{nlags},$$

where N is `nrows(x)` and $\overline{X} = \frac{1}{N}\sum_{t=0}^{N-1} X_t$ is the sample mean. The sums of lagged products are computed using DFT's of length `L`, which must be at at least `nlags + nrows(x)`. If `L` is omitted and variable `S` is defined, the value of `S` is used for `L`. If `S` is not defined, the smallest power of 2 exceeding `nrows(x) + nlags` is used for `L`. If the second argument is omitted, `nrows(x)-1` is used for `nlags`. Generally it is convenient to decide on the length of the DFT's to be used and define variable `S`.

```
Cmd> getmacros(autocov,quiet:T)

Cmd> acf <- autocov(x,60) # autocovariances up to 60 lags

Cmd> tsplot(acf,0,impulse:T,ymin:-max(acf),xlab:"Lag in Years",\
title:"Autocovariance function for AR(2)",ylab:"Auto Covariance")
```



Keyword phrase `impulse:T` adds the vertical lines between the ACF and the ACF = 0 line. See Sec. 8.5.2.

5.4.4 **Removing a polynomial trend – detrend** The first step in the analysis of a time series is often the removal of the sample mean or a trend function. If `x` is a matrix and `k` is an integer, column j of `detrend(x,k)` is the j^{th} column of `y` minus the best

fitting (in the least squares sense) polynomial of degree k. If k is omitted, its default value is 1 and a linear trend is subtracted from each column.

Caution: For some older versions of detrend, the default degree was 0 rather than 1.

```
Cmd> getmacros(detrend,quiet:T)

Cmd> x1 <- vector(7,9,8,2,3,6,14,2,4,9) # short time series

Cmd> detrend(x1) # remove linear trend, default degree 1
(1)      0.38182       2.4303       1.4788      -4.4727      -3.4242
(6)     -0.37576       7.6727      -4.2788      -2.2303       2.8182

Cmd> detrend(x1,0) # subtract mean, degree 0 polynomial
(1)          0.6          2.6          1.6         -4.4         -3.4
(6)         -0.4          7.6         -4.4         -2.4          2.6

Cmd> detrend(x1,2) # remove quadratic trend; 2nd degree polynomial
(1)     -0.89091       2.0061       1.6909      -3.8364      -2.5758
(6)      0.47273       8.3091      -4.0667      -2.6545       1.5455
```

5.4.5 **Using tapers (data windows) – costaper and compza** Frequency domain estimation is based on computing Fourier transforms to attempt to extract from a time series what is "going on" at each frequency. From Fourier transforms you can build estimates of a spectrum or a cross spectrum. An inevitable problem is "leakage" whereby what is "going on" at frequency f affects the Fourier transform at other frequencies distant from f. This can introduce bias in estimated spectra and cross spectra and can result in excessive correlation between estimates at different frequencies.

Although some leakage is inevitable, it can be minimized by appropriate "pre-processing" of a time series before computing a Fourier transform.

First, almost always, you should subtract an estimate of the mean $\mu_t = E[X_t]$ of the series. When you can assume the mean is constant, you can estimate μ_t by the sample mean $\overline{X} = \frac{1}{N}\sum_{t=0}^{N-1} X_t$. If not, then it is often sufficient to estimate μ_t by a polynomial fitted by least squares. More sophisticated methods of estimation may sometimes be needed. Thus the basic input to frequency domain analysis is usually a series of residuals $X_t - \hat{\mu}_t$, where $\hat{\mu}_t$ estimates μ_t. For notational simplicity, we will refer to these residuals simply as X_t.

Secondly, it is important to "taper" the residuals by multiplying them by a *tapering function* or *data window*. A tapering function is a sequence $a_0, a_1, \ldots a_{N-1}$, where the "tails" usually taper smoothly to 0. Define the (continuous) *modified* or *tapered* Fourier transform of a series $\{X_t\}$ as the CFT of the tapered series

$$\hat{X}_A(f) = \frac{1}{\sqrt{K_A}}\sum_{t=0}^{N-1} a_t X_t e^{-i2\pi tf}, \; K_A = \sum a_t^2 .$$

$\hat{X}(f)/\sqrt{N}$, computed from the ordinary Fourier transform, is a modified Fourier transform with constant weights $a_t \equiv 1$ (which do not taper at all) and with $K_A = N$. If $\{a_t\}$ is chosen appropriately, there will be much less leakage in $\hat{X}_A(f)$ than in $\hat{X}(f)$. It is

the lack of any tapering in the ordinary Fourier transform $\hat{X}(f)$ that is the root cause of leakage in frequency domain estimates based on it.

Typically a taper has a maximum near $(N-1)/2$ and is symmetric in that its tails near $t = 0$ and $t = N-1$ are mirror images of one another, that is $a_{N-1-t} = a_t$. Often it is constant over a large percentage of its range. One popular taper is a cosine taper defined as

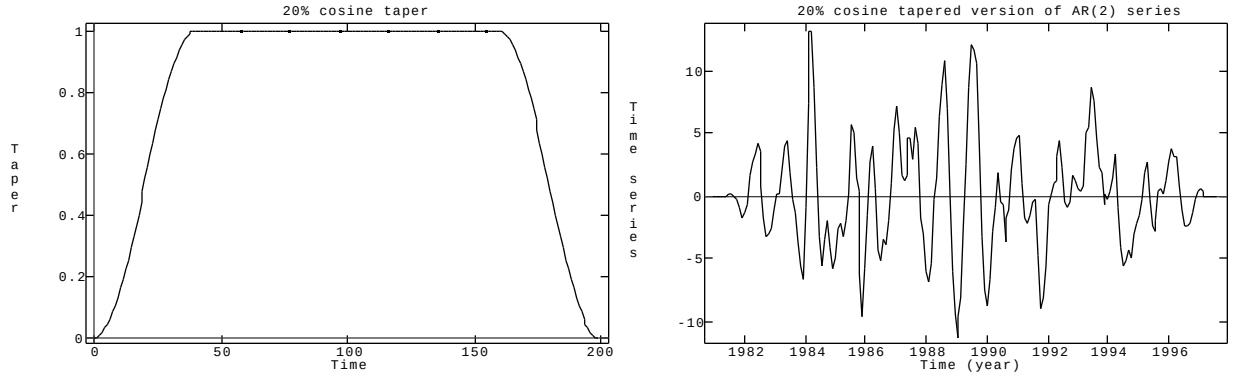
$$a_t = \begin{cases} \frac{1}{2}(1-\cos((t+\frac{1}{2})/m)\pi) & t = 0,1,\ldots, m-1 \\ 1 & t = m,\ldots, N-m-1, \\ \frac{1}{2}(1-\cos((N-t-\frac{1}{2})/m)\pi) & t = N-m,\ldots, N-1 \end{cases}$$

where m = ⌈αN⌉ = smallest integer $\geq \alpha N$, with $0 \leq \alpha \leq .5$. This is said to be a 100α% taper , tapering about 100α% of the entire series at each end. (Some sources call it a $100\times 2\alpha$ % taper because appoximately $2\alpha N$ elements are modified.) When option `angles` has been set to `"cycles"` (Sec. 8.1), $\{a_t\}$ might be computed in MacAnova by

```
Cmd> a <- vector(.5*(1-cos(.5*(run(0,m-1)+.5)/m)),rep(1,N-2*m),\
     .5*(1-cos(.5*(N-run(N-m,N-1)-.5)/m)))
```

Macro `costaper` does this computation. Specifically, `costaper(n,alpha)` computes a cosine taper of length `n` with `100*alpha` percent tapering on either end.

```
Cmd> getmacros(costaper,quiet:T)

Cmd> tsplot(costaper(200,.2),0,1,xlab:"Time",ylab:"Taper",\
title:"20% cosine taper")

Cmd> tsplot(costaper(200,.2)*detrend(x,0),1981,xlab:"Time (year)",\
title:"20% cosine tapered version of AR(2) series")
```



`compza(x,alpha:A,degree:d,S:s)` computes a modified Fourier transform $\hat{X}_A(f)$ of $\{X_t - \bar{X}\}$ at frequencies $f = 0, 1/S, \ldots, (S-1)/S$, returning a structure with five components, `za` (modified Fourier transform), `n` (`nrows(x)`), `ka` (K_A), `alpha` and `degree`, where S = `s`. Residuals from a `d` degree polynomial fitted by least squares are tapered by a cosine taper with α = `A`. All the keyword phrases are optional. The default values for `A` and `d` are 0.1 and 0, respectively. The default value for `s` is the value of variable `S` if it defined, or `2*nrows(x)` otherwise.

For example, `compza(x,alpha:.1,degree:2, S:500))`, computes the modified Fourier transform at 500 frequencies with 10% cosine tapering after removing a

quadratic trend (polynomial degree 2) from the columns of x.

```
Cmd> getmacros(compza,quiet:T)

Cmd> compza(x1,alpha:.1,S:20)
component: za
 (1)      -0.5488      0.47102       1.0048       2.9934       3.4881
 (6)      -5.0421       -1.948       2.4692      -3.2995     -0.37712
(11)        3.087       3.3745      -1.1749      -1.3944       3.9707
(16)      -4.2189      -4.8188      0.82104      0.65077      0.95695
component: n
(1)            10
component: ka
(1)           8.5
component: alpha
(1)           0.1
component: degree
(1)             0
```

Let's check the results.

```
Cmd> alpha <- .1; ka <- sum(costaper(10, alpha)^2); ka
(1)           8.5

Cmd> rft(padto(costaper(10, alpha)*detrend(x1,0),20))/sqrt(ka)
 (1)      -0.5488      0.47102       1.0048       2.9934       3.4881
 (6)      -5.0421       -1.948       2.4692      -3.2995     -0.37712
(11)        3.087       3.3745      -1.1749      -1.3944       3.9707
(16)      -4.2189      -4.8188      0.82104      0.65077      0.95695
```

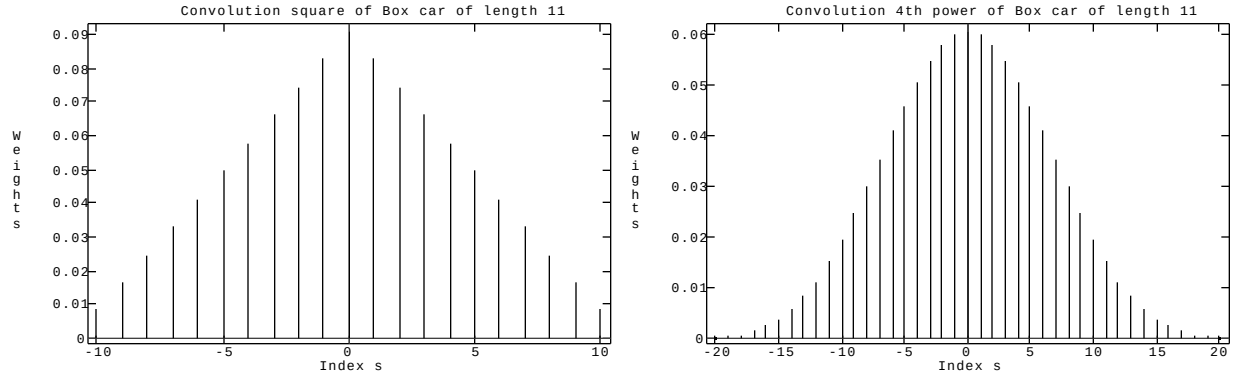
5.4.6 **Smoothing periodograms – compfa and spectrum** A popular Fourier transform based method to estimate the spectrum $S_{XX}(f)$ of a covariance stationary time series $\{X_t\}$ is to smooth its periodogram $I_{XX}(f) = \frac{1}{N}\left|\hat{X}(f)\right|^2$, computed at S equally spaced frequencies. This is done by convolving $I_{XX}(f)$ with a sequence of weights $\{w_s\}$ where $\sum_s w_s = 1$. Usually $w_s \geq 0$ and $w_{-s} = w_s$. Usually an estimate of a mean or trend is subtracted from X_t before computing $I_{XX}(f)$. Sometimes the periodogram is defined to be $2I_{XX}(f)$. It is often a good idea to have $S \approx 2N$.

Because of the poor leakage properties of the unmodified Fourier transform $\hat{X}(f)$ it is better to smooth a *modified periodogram* $I_{AXX}(f) = \left|\hat{X}_A(f)\right|^2$ based on the modified Fourier transform (Sec. 5.4.5).

Typically the weights satisfy $w_s = 0$ for $|s| > m$, have a maximum at $s = 0$ and decline monitonically for $|s| = 1, ..., m$. The case when the weights are equal ($w_{-m} = w_{-m+1}$ $... = w_0 = ... w_m = 1/(2m + 1)$) is some times referred to as "boxcar" smoothing. A variety of other smoothing weights can be obtained by convolving boxcar smoothers with themselves. For example, convolving a boxcar of length $n = 2m + 1$ with itself (the "convolution square") produces a "triangular" smoother of length $2n - 1 = 4m + 1$ with weights increasing linearly for $s = -2m, -2m + 1, ..., 0$ and then decreasing linearly for $s = 0, 1, ..., 2m$. Convolving such a boxcar with itself four times (the "4th convolution power") produces a bell shaped smoother of length $4n - 3 = 8m + 1$

whose weights are non-zero for $s = -4m, -4m+1, ..., 4m-1, 4m$.

```
Cmd> m <- 5;boxcar <- rep(1/(2*m+1), (2*m+1)) # boxcar of length 11
Cmd> triangle <- convolve(boxcar,padto(boxcar,4*m+1)) # length 21
Cmd> bell <- convolve(triangle,padto(triangle,8*m+1)) # length 41
Cmd> plot(run(-2*m,2*m),triangle,impulse:T,ymin:0,xlab:"Index s",\
       ylab:"Weights",\
       title:"Convolution square of Box car of length 11")

Cmd> plot(run(-4*m,4*m),bell,impulse:T,ymin:0,xlab:"Index s",\
       ylab:"Weights",\
       title:"Convolution 4th power of Box car of length 11")
```



We need to use `padto()` in computing the convolutions because `convolve()` does circular convolutions. See Sec. 5.2.8.

When $w_s = 0$ for $|s| > m$, `iaxx` is a (modified) periodogram in Real or Complex (not Hermitian) form and `wts` is a REAL vector of length $2m+1$ with `wts[1]` = w_{-m}, `wts[2]` = w_{-m+1},..., `wts[m+1]` = w_0, ..., `wts[2*m+1]` = w_m, you can smooth `iaxx` by `rotate(convolve(wts,iaxx),-m)`. The use of `rotate()` compensates for the fact `wts[t]` = w_{t-1-m}. Here we do *not* use `padto()` to pad with zeros because circular convolution is exactly what is needed because the periodogram is a periodic function.

In almost any estimation situation, you should be interested in the precision of an estimator. One measure of precision for a positive estimator is its *equivalent degrees of freedom* or EDF. If $W \geq 0$ is a random variable, $\mathrm{EDF}[W] \equiv 2\mathrm{E}[W]^2/\mathrm{V}[W]$. A large value of EDF[$W$] means the standard deviation of W is small relative to its mean and hence the relative precision of W as an estimate of E[W] is high.

When W is distributed as a multiple of χ^2, say $W = \tau^2 \frac{\chi^2_d}{d}$, then $\mathrm{E}[W] = \tau^2$ and $\mathrm{EDF}[W] = d$. A common approximation to the distribution of a random variable $W \geq 0$ is to treat it as if it were $\tau^2 \frac{\chi^2_d}{d}$ with $\tau^2 = \mathrm{E}[W]$ and $d = \mathrm{EDF}[W]$, so that $\mathrm{E}[W] = \mathrm{E}[\tau^2 \frac{\chi^2_d}{d}]$ and $\mathrm{V}[W] = \mathrm{V}[\tau^2 \frac{\chi^2_d}{d}]$. This approximation is widely used in spectrum estimation.

In the context of spectrum estimation, the more a periodogram or modified periodogram is smoothed to compute an estimated spectrum $\hat{S}_{XX}(f)$, the larger is EDF[$\hat{S}_{XX}(f)$]. This results in a smaller variance but usually a larger bias.

Under broad conditions, when $\hat{S}_{XX}(f)$ is computed by smoothing a modified periodogram computed at S equally spaced frequencies, EDF[$\hat{S}_{XX}(f)$] is approximately $\frac{2R_A N}{S\sum w_s^2}$, where $\{w_s\}$ are the smoothing weights and $R_A = \frac{\{\sum a_s^2\}^2}{N\sum a_s^4} \le 1$. $R_A = 1$ when there is no tapering. For a cosine taper, $R_A = (1 - 1.25m/N)^2/(1 - 1.453125m/N)$, where $m = \text{é}\alpha N\text{ù}$.

An important characteristic of the estimate is its *bandwidth*, the effective frequency range over which appreciable smoothing occurs. The greater the bandwidth, the more stable (larger EDF) is the resulting estimate, but the greater the potential for bias because of the smoothing. Conversely, the smaller the bandwidth, the less the bias, at the cost of decreased EDF in the estimate.

A common definition of the bandwidth associated with a periodogram computed at S frequencies and smoothed by weights $\{w_s\}$ with $\sum_i w_i = 1$ is $B = \frac{1}{S\sum_i w_i^2}$, in units of cycles per Δ_t. When $\{w_s\}$ is a boxcar of length n, $B = n/S \approx .5n/N$ when $S \approx 2N$. For the 4th power of a boxcar of length n,

$$\sum_i w_i^2 = (151n^7 + 70n^5 + 49*n^3 + 45n)/(315n^8) \approx .48/n$$
$$B = (2.086n/S)(1 - .464/n^2 + O(1/n^4))$$

When the spectrum is sufficiently smooth, the EDF of a smoothed modified periodogram with bandwidth B at a frequency f between $B/2$ and $1/2 - B/2$ cycles is approximately $2R_A BN$. At 0 and 1/2 cycles, EDF $\approx R_A BN$.

`compfa(x,edf,alpha:A,degree:d,S:s)`, where `edf` ≥ 2, computes smoothed modified periodograms as estimates $\hat{S}_{X_jX_j}(f)$ of the spectra of time series in the columns of `REAL` matrix `x` with N rows. Residuals from a polynomial of degree `d` fit by least squares are tapered using a cosine taper tapering 100A% on each end. The modified periodograms are smoothed with the 4th convolution power of a boxcar of the right length so that estimated spectra have EDF $\approx$ `edf` when $S_{XX}(f)$ doesn't vary much over the width of the smoothing weights. By default, the actual EDF computed from the formula is printed. It is usually slightly larger than the value of `edf`.

`compfa` returns a matrix with the same number of columns as `x`, but with S = `s` rows. Column j of the result is a Real series containing the estimated spectrum $\hat{S}_{X_jX_j}(f)$ of column j of `x` computed at S frequencies $f = 0, 1/S, 2/S, ..., (S-1)/S$.

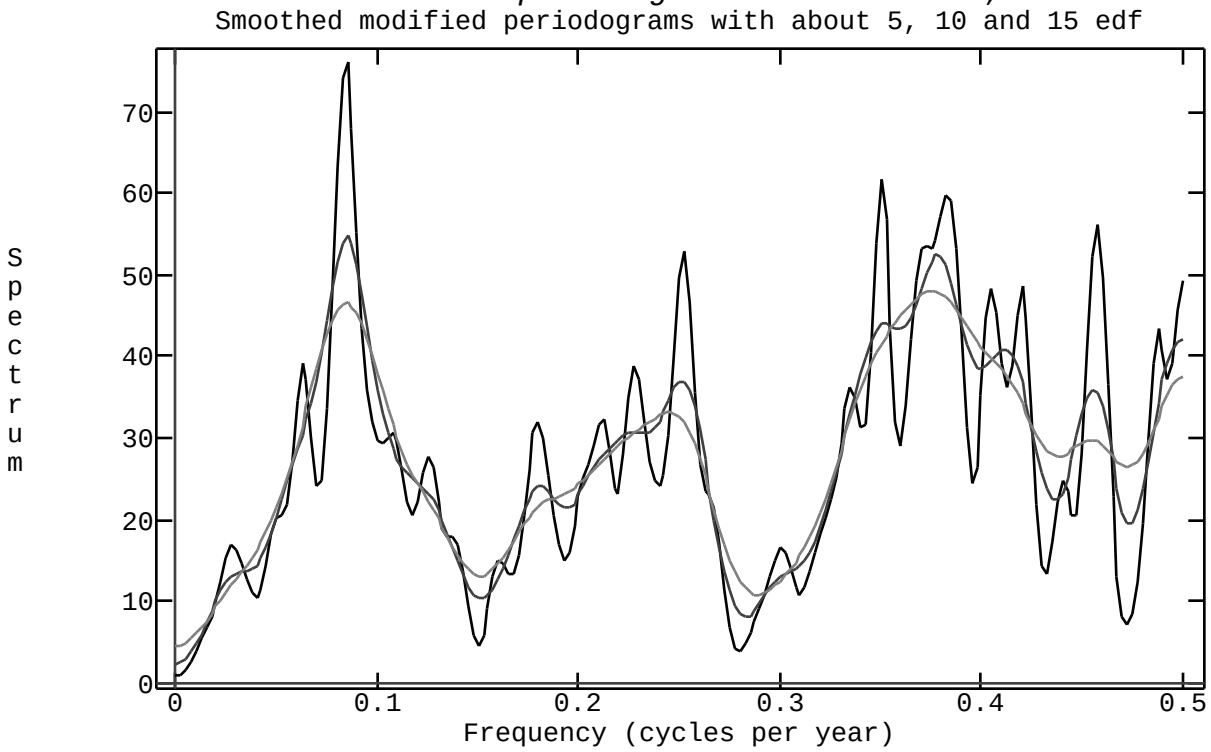
All the keyword phrases are optional. The default values for `A` and `d` are .10 and 0, respectively. The default value for `s` is the value of variable `S` if it exists or $2N$. If N has any prime factor > 29, `S` must be set appropriately so that its value doesn't have such factors.

```
Cmd> getmacros(compfa,quiet:T)

Cmd> sazz <- compfa(z,15,alpha:.1) # note explanatory comment
rep(1/9,9)^*4 smoother with 16.7 edf, S = 400
```

You can suppress the descriptive comment by keyword phrase `quiet:T`.

A value of `edf` with 0 < `edf` < .5 is also legal. It this case `edf` is interpreted to specify a desired bandwidth B = `edf` in cycles. The actual EDF aimed for is then taken to be $2BN$. `edf` = 0 or 2 results in no smoothing.

```
Cmd> ffplot(hconcat(compfa(z,5,alpha:.1,quiet:T),\
compfa(z,10,alpha:.1,quiet:T),compfa(z,15,alpha:.1,quiet:T)),\
xlab:"Frequency (cycles per year)",ylab:"Spectrum",\
title:"Smoothed modified periodograms with about 5, 10 and 15 edf")
```



Macro `spectrum` is an alternative to `compfa()` which has fewer options and lacks the capability to taper or remove a polynomial trend. There are more choices for smoothing but they are specified differently.

`spectrum(x,m,p)` computes the smoothed *unmodified* periodograms $\hat{S}_{XX}(f)$ of each column of REAL matrix `x` with N rows. The weights $\{w_s\}$ are computed as the `p`th convolution power of a boxcar of length `m`. `spectrum(x,m)` is equivalent to `spectrum(x,m,4)`. No tapering is done (see Sec. 5.4.5) and only the sample mean of each column is subtracted before computing the Fourier transforms.

When `m` = 1, the columns of the output are the unsmoothed periodograms $I_{XX}(f)$.

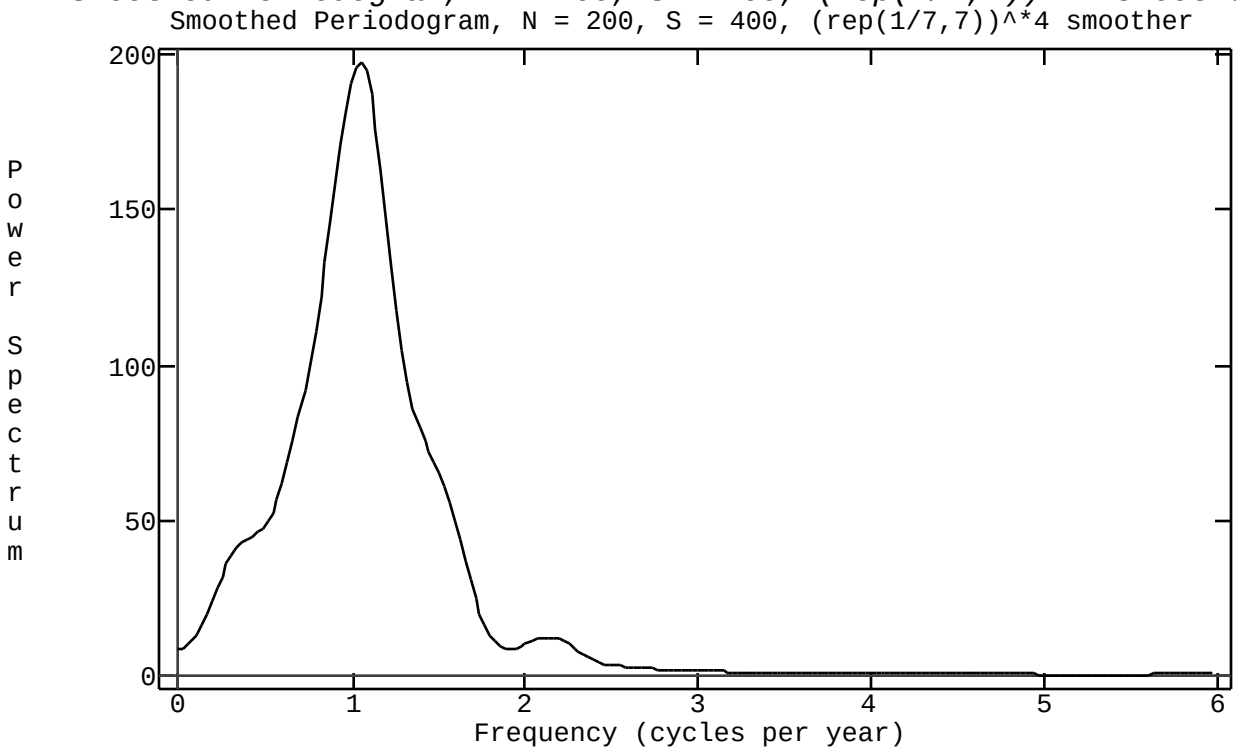
`spectrum` returns a matrix with the same number of columns as `x`, but with S rows where S is the value of variable `S` if it exists or $2N$ otherwise. If N has any prime factor > 29, `S` must be set appropriately so that its value doesn't have such factors.

Column j of the result is a Real series containing the estimated spectrum $\hat{S}_{X_jX_j}(f)$ of

column j of x computed at frequencies $f = 0, 1/S, 2/S, ..., (S–1)/S$. Because $\hat{S}_{XX}(f)$ is symmetrical about $f = 0$ and periodic with period 1, $\hat{S}_{XX}(j/S) = \hat{S}_{XX}(–j/S) = \hat{S}_{XX}(S – j/S)$, $j = 1, ..., S–1$.

`spectrum(costaper(nrows(x),A)*detrend(x,d),m,p)` does the same, working with tapered residuals from a polynomial trend, just as does `compfa`.

```
Cmd> getmacros(spectrum,quiet:T)
Cmd> S <- 2*nrows(x) # set number of frequencies
Cmd> sxxhat <- spectrum(x,7)
Cmd> ffplot(sxxhat,ymin:0,xlab:"Frequency (cycles per year)",\
ylab:"Power Spectrum",title:\
"Smoothed Periodogram, N = 200, S = 400, (rep(1/7,7))^*4 smoother")
```

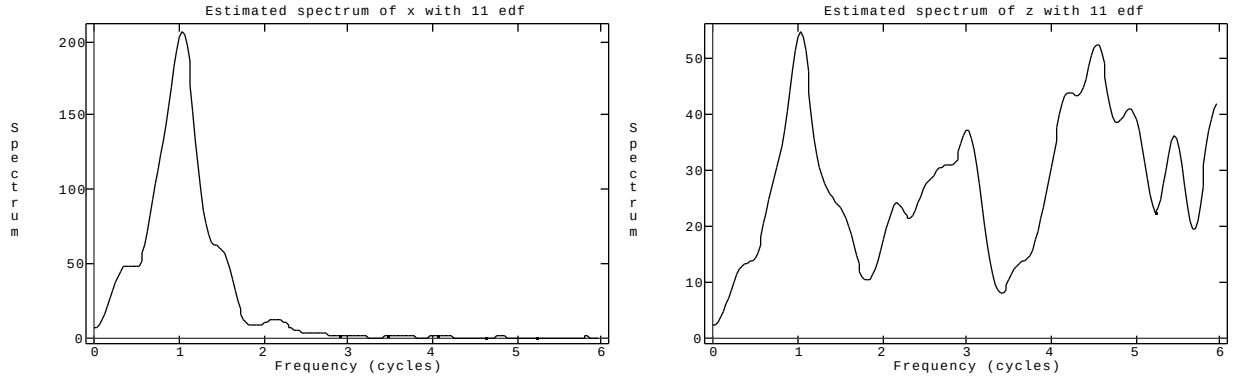


5.4.7 **Smoothing cross-periodograms – compfa and crsspectrum** Analogously to the spectrum, the cross spectrum of two time series $\{X_t\}$ and $\{Y_t\}$ is sometimes estimated by smoothing their *cross-periodogram* $I_{XY}(f) = N^{-1}\hat{X}(f)\overline{\hat{Y}(f)}$ but it is usually better to smooth their modified cross-periodogram $I_{AXY}(f) = \hat{X}_A(f)\overline{\hat{Y}_A(f)}$. As before, the time series should be detrended before tapering. Macro `compfa` can also be used for estimating cross spectra by including keyword phrase `cross:T` as an argument.

`compfa(x,edf,cross:T,alpha:A,degree:d,S:s)` computes estimates of the spectrum of each column of x and estimates of the cross spectra between any two columns, all at frequencies $f = 0, 1/S, 2/S, ..., (S–1)/S$. When x is a N by q matrix, the result is a S by $q(q+1)/2$ matrix, where S is as in Sec. 5.4.6. The first q columns of the result are the same as would be produced by `compfa` without `cross:T`, namely

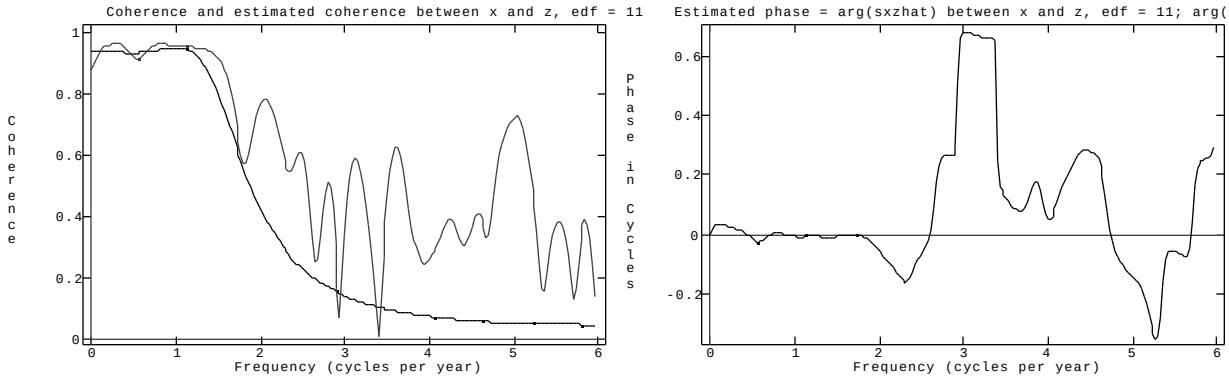
Real series containing the estimated spectra $\hat{S}_{X_jX_j}(f)$ of the columns of x. The remaining columns of the result are the estimated cross spectra $\hat{S}_{X_iX_j}(f)$, in Hermitian form (Sec. 5.2.3), between columns 1 and 2, columns 1 and 3, ..., columns 1 and q, columns 2 and 3, ..., columns 2 and q, ..., and columns q–1 and q.

```
Cmd> shat <- compfa(hconcat(x,z),10,alpha:.1,cross:T,S:400)
rep(1/6,6)^*4 smoother with 11.1 edf, S = 400

Cmd> list(shat) # Columns are sxxhat, szzhat, sxzhat
shat            REAL   400   3

Cmd> ffplot(shat[,1],ymin:0,\
    lab:"Frequency (cycles)",ylab:"Spectrum",\
    title:"Estimated spectrum of x with 11 edf")

Cmd> ffplot(shat[,2],ymin:0,\
    xlab:"Frequency (cycles)",ylab:"Spectrum",\
    title:"Estimated spectrum of z with 11 edf")
```




```
Cmd> sxzhat <- shat[,3] # Hermitian form of cross spectrum

Cmd> coherhat <- hreal(hpolar(sxzhat))/sqrt(shat[,1]*shat[,2])

Cmd> phasexz <- himag(hpolar(sxzhat)) # arg(sxzhat)

Cmd> #See Sec. 5.2.4 for hreal(), himag(), and hpolar()
```

```
Cmd> ffplot(hconcat(coher,coherhat),ymin:0,ymax:1,\
xlab:"Frequency (cycles per year)",ylab:"Coherence",\
title:"Coherence and estimated coherence between x and z, edf = 11")

Cmd> ffplot(phasexz,xlab:"Frequency (cycles per year)",\
ylab:"Phase in Cycles",\
title:"Estimated phase = arg(sxzhat) between x and z, edf = 11;
arg(sxz) = 0")
```



In these last two plots, the quantities plotted are the estimated coherence $\frac{\left|\hat{S}_{XY}(f)\right|}{\sqrt{\hat{S}_{XX}(f)\hat{S}_{YY}(f)}}$ and the *phase* $\hat{\varphi}_{XY}(f)$ of $\hat{S}_{XY}(f)$, where $\hat{S}_{XY}(f) = \left|\hat{S}_{XY}(f)\right| e^{i\hat{\varphi}_{XY}(f)}$. Also plotted are the true coherence and phase (the true phase is 0 for all *f*).

The lines computing `coherhat` and `phasexz` are a little tricky. They work because `hpolar()` returns its result as a pseudo Hermitian series whose real and imaginary parts are the amplitude and phase, respectively. See Sec. 5.2.3, 5.2.4.

`crsspectrum(x,m,p)` computes output similar to `compfa` with argument `cross:T` except that the smoothing weights are the p^{th} convolution power of a boxcar of length `m`. `crsspectrum(x,m)` is equivalent to `crsspectrum(x,m,4)`. The sample means are removed from the columns of `x` but no other detrending is done and the residuals from the mean are not tapered. As with spectrum, you can detrend and taper by `crsspectrum(costaper(nrows(x),A)*detrend(x,d),m,p)`. Here is how you could get the previous results with `crsspectrum`.

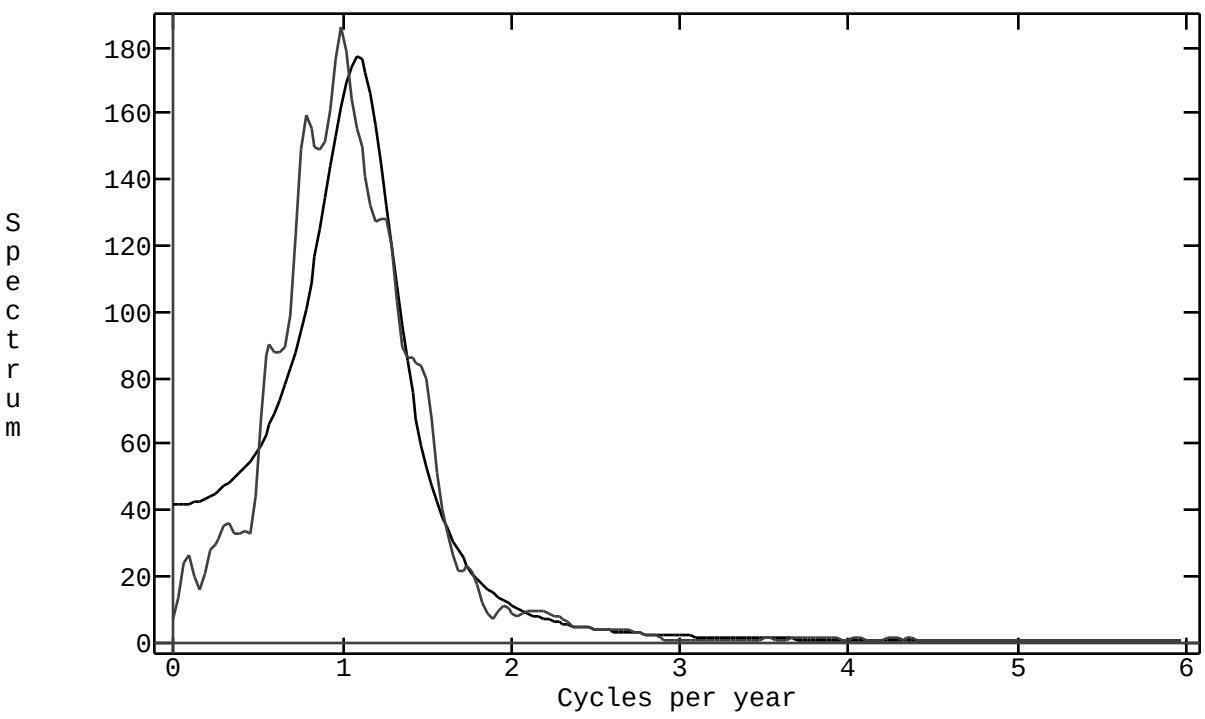
```
Cmd> getmacros(crsspectrum,quiet:T)

Cmd> S <- 400; shat1 <- crsspectrum(costaper(200,.1)*hconcat(x,z),6)

Cmd> # This does same tapering, smoothing as was done by compfa

Cmd> list(shat1) # columns are sxxhat, szzhat, sxzhat as for compfa
shat1           REAL   400   3
```

5.4.8 **Multi-taper spectrum estimation – multitaper** A fairly recent approach to spectrum estimation averages several modified periodograms, computed with a set of distinct tapers that (a) taper much more severely than a cosine taper, even with $\alpha = .5$, and (b) are orthogonal in the sense that their summed cross products are zero. The tapers used are *discrete prolate spheroidal sequences* (DPSS) which can be computed as eigenvectors of certain tridiagonal matrices. A particular estimate is determined by (i)

the bandwidth W associated with the taper, and the number K of orthogonal tapers to use. Percival and Walden (1993) provide a thorough discussion of multitaper spectrum estimation, including the computation of DPSS tapers. Macro `dpss` in file Tser.mac uses function `trideigen()` (Sec. 6.2.2) to compute these tapers.

`multitaper(x,W,K,degree:d,S:s)` computes a multitaper spectrum estimate of the time series in REAL vector `x`. `K` must be a positive integer and `W` is a positive REAL scalar < .5/`DELTAT` specifying the taper bandwidth in cycles per unit time. The estimate is computed from residuals (from a polynomial trend of degree `d` fitted by least squares) at S = `s` frequencies 0, 1/S, ... (S–1)/S. The result is a Real series. All the keyword phrases are optional. The default value for `d` is 0, and the default for `s` is the value of `S` when `S` is defined or `2*nrows(x)` otherwise. `multitaper` makes use of macro `dpss`, attempting to read it in if it is not available.

```
Cmd> getmacros(dpss,multitaper,quiet:T)

Cmd> ffplot(hconcat(sxx,multitaper(x,.05/DELTAT,5)),\
xlab:"Cycles per year",ylab:"Spectrum",\
title:"Spectrum and multi-taper spectrum estimate, W=.6 cycles per
year, K = 5")
```

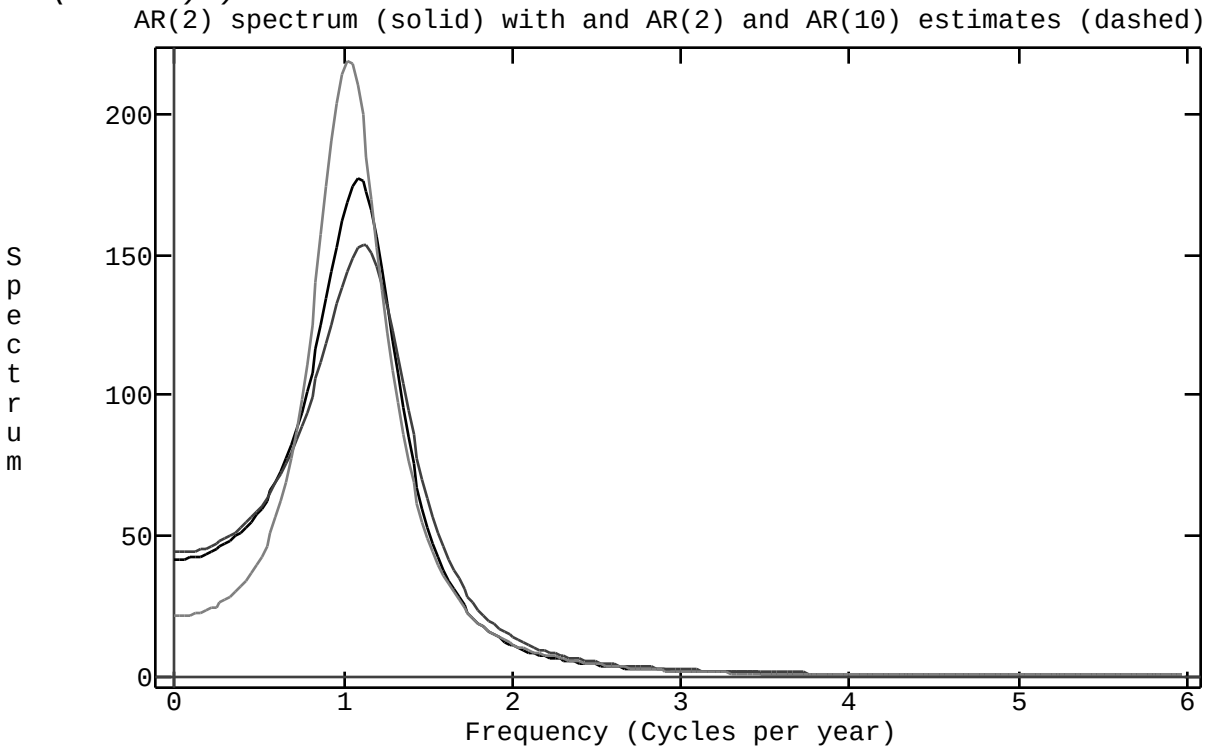


5.4.9 **Autoregressive spectrum estimation – arspectrum and burg** Yet another approach to spectrum estimation is to require the estimate to be a member of a flexible family of spectra determined by a relatively small set of parameters. One such family are the spectra $S_{XX}(f) = S_{XX}(f;\varphi_1, \quad ,\varphi_p,\sigma^2) = \dfrac{\sigma^2}{\left|1-\sum_{s=1}^{p}\varphi_s e^{-i2\pi sf}\right|^2}$ of a purely AR(p) time series (see Sec. 5.3.1), with coefficients φ_1, ..., φ_p and innovation variance σ^2. An estimate takes the form $\hat{S}_{XX}(f) = S_{XX}(f;\hat{\varphi}_1, \quad ,\hat{\varphi}_p,\hat{\sigma}^2)$, where $\hat{\varphi}_1$, $\quad ,\hat{\varphi}_p$ and $\hat{\sigma}^2$ are estimates of φ_1, ..., φ_p and σ^2. MacAnova provides two macros based on this idea, implementing quite different algorithms to estimate φ_1, ..., φ_p.

`arspectrum(x,p,s)` solves the Yule-Walker equations $r_s = \sum_{k=1}^{p}\varphi_k r_{s-k}$, $s = 1, 2, \ldots, p$, for φ_k, where $r_s = \dfrac{C_{|s|}}{C_0}$ is the sample autocorrelation function (see Sec. 5.4.3), and estimates σ^2 as $\hat{\sigma}^2 = C_0\prod_{k=1}^{p}(1-\hat{\varphi}_{kk}^2)$, where $\hat{\varphi}_{kk}$ is the estimated k^{th} partial autocorrelation computed from r_s, $s = 1,\ldots,k$ (see Sec. 5.3.4). The result is a structure with components `phi` (estimates $\hat{\varphi}_k$), `var` ($\hat{\sigma}^2$) and `spectrum` ($\hat{S}(f)$), computed at frequencies 0, $1/S$, ..., $(S-1)/S$, where S = `s.arspectrum(x,p)` does the same, with $S = 2^m$, where m is the smallest integer for which $2^m >$ `2*nrows(x)`.

```
Cmd> getmacros(arspectrum,quiet:T)

Cmd> arspect <- arspectrum(x,2,400) # use 400 frequencies

Cmd> compnames(arspect) # arspect has 3 components
(1) "phi"
(2) "var"
(3) "spectrum"

Cmd> arspect[run(2)] # first two components
component: phi
(1)       1.3658     -0.69382
component: var
(1)       4.7358

Cmd> sxx2 <- arspect$spectrum

Cmd> sxx10 <- arspectrum(x,10,400)$spectrum # now fit AR(10)
```

```
Cmd> ffplot(hconcat(sxx,sxx2,sxx10),\
xlab:"Frequency (Cycles per year)",ylab:"Spectrum",\
title:"AR(2) spectrum (solid) with and AR(2) and AR(10) estimates
(dashed)")
```



`burg(y,p,degree:d,S:s)` uses an algorithm due to Burg (Kirk, Rust, Van Winkle 1979) which estimates the partial autocorrelations $\varphi_{11},\ldots,\varphi_{pp}$ one by one directly from the data, minimizing at each stage a certain sum of squares. These are used to compute estimates of φ_k and σ^2. The result is a structure of the same form as returned by `arspectrum`. The spectrum is estimated at S = `s` equally spaced frequencies from `detrend(y,d)`.

The keyword phrases are optional. The default value of `d` is 0 and the default value of `s` is the value of `S` if it is defined or `nrows(x)+p` if not.

```
Cmd> getmacros(burg,quiet:T)

Cmd> burgspect<-burg(x,2,S:400) # use 400 frequencies

Cmd> burgspect[run(2)] # first two components
component: var
(1)       4.0165
component: phi
(1)       1.4143     -0.74251

Cmd> sxx2 <- burgspect$spectrum

Cmd> sxx10 <- burg(x,10,S:400)$spectrum # now use p = 10
```

```
Cmd> ffplot(hconcat(sxx,sxx2,sxx10),\
xlab:"Frequency (Cycles per year)",ylab:"Spectrum",\
title:"AR(2) spectrum (solid) with Burg AR(2) and AR(10) estimates
(dashed)")
```