3 Interpolation and extrapolation

Functions:

polint: interpolation, polynomial
ratint: interpolation, rational function
spline: interpolation, construct a cubic spline
splint: evaluate a cubic spline
locate: search an ordered table, bisection
hunt: search an ordered table, correlated calls
polcoe: polynomial coefficients from a table of values
polcof: polynomial coefficients from a table of values
polin2: interpolation, two dimensional polynomial
bcucof: interpolation, two dimensional construct bicubic
bcuint: interpolation, two dimensional evalute bicubic
splie2: interpolation, two dimensional, construct two dimensional spline
splin2: interpolation, two dimensional, evaluate two dimensional spline

---

polint

Description:

Given lists xa and ya this function uses a polynomial of degree $n-1$ such that $p(xa_i) = ya_i$ to produce an interpolated value $y = p(x)$.

polint(xa, ya, x) $\longrightarrow$ [y, dy]

Where:

xa is a list of real numbers representing the independent variables,
ya is a list of real numbers representing the dependent variables,
x is a real number being the point at which the interpolation is required,
y is a real number representing the interpolated value,
dy is a real number giving an error estimate for the interpolation.

Example:

In this example a polynomial interpolant to the exponential function on [0,1] is computed using 10 function value points. The result is checked at 0.05.

```
-> xa : [0.05,0.15,0.25,0.35,0.45,0.55,0.65,0.75,0.85,0.95];
-> set ya to the list of values such that ya(i)=exp(xa(i)) #
-> ya : map(exp,xa)?
[1.0512711, 1.1618342, 1.2840254, 1.4190675, 1.5683122, 1.7332530,
1.9155408, 2.1170000, 2.3396468, 2.5857096]
-> polint(xa,ya,0.05)?
[1.0512711, 0.0000000]
-> Compare with: #
-> exp(0.05)?
1.0512711
-> polint(xa,ya,0.5)?
[1.6487213, -0.0000000]
-> Compare with: #
-> exp(0.5)?
1.6487212
```

ratint

Description:

Given lists xa and ya of length n this function uses a diagonal rational function such that $r(xa_i) = ya_i$ to produce an interpolated value $y = r(x)$.

ratint(xa, ya, x) $\longrightarrow$ [y, dy]

Where:

xa is a list of real numbers representing the independent variables,
ya is a list of real numbers representing the dependent variables,
x is a real number being the point at which the interpolation is required,
y is a real number representing the interpolated value,
dy is a real number giving an error estimate for the interpolation.

Pre-variables:

ratint_tiny [1.0d-25] a small number to prevent 0 over 0 occuring.

Example:

In this example a rational function interpolant is computed for a function with an infinite singularity at the point 1.0 using `ratint` and 6 points.

```
-> g(s) := s*exp(-s)/((s-1.0)^2 + 1.0);
-> x : [0.1,0.3,0.5,0.7,0.9,1.1];
-> y : map(g,x)?
[0.0499910, 0.1491580, 0.2426123, 0.3189080, 0.3622898, 0.3625329]
-> Diagonal rational function interpolation #
-> for i from 4 to 10 do
  sprog(xx : 0.2*i,
    print(["x,         interp,     accuracy,    actual"]),
    print([xx, ratint(x,y,xx), g(xx)]));
["x,         interp,     accuracy,    actual"]
[0.8000000, [0.3456280, -0.0001988], 0.3456377]
["x,         interp,     accuracy,    actual"]
[1.0000000, [0.3679053, 0.0018816], 0.3678795]
["x,         interp,     accuracy,    actual"]
[1.2000000, [0.3473104, 0.0009797], 0.3475318]
["x,         interp,     accuracy,    actual"]
[1.4000000, [0.2956437, 0.0083734], 0.2976170]
["x,         interp,     accuracy,    actual"]
[1.6000000, [0.2313336, 0.0261430], 0.2375253]
["x,         interp,     accuracy,    actual"]
[1.8000001, [0.1688408, 0.0539588], 0.1814256]
["x,         interp,     accuracy,    actual"]
[2.0000000, [0.1152596, 0.0882240], 0.1353353]
```

spline

Description:

Given lists x and y of numbers of length n representing a tabulated function $f(x_i) = y_i$, with the values of x strictly increasing and given values $yp1$ and ypn of the derivatives of the interpolating function at x_1 and x_n respectively, this function returns a list $y2$ of length n representing the second derivative values of the cubic spline interpolant of f at the points x.

spline(x, y, yp1, ypn) $\longrightarrow$ y2

Where:

x is a real list representing the independent variables,
y is a real list representing the dependent variables,
yp1 is a real number representing the derivative of the interpolant at x_1,
ypn is a real number representing the derivative of the interpolant at x_n,
y2 is a real list representing the interpolated values at the points x.

Example:

In this example the `sin` function is interpolated over the range $[0, \pi]$ using the function `spline` and 10 points. The second derivatives of the cubic spline interpolant are computed and compared with the true values.

```
-> Second-derivatives for sin(x) from 0 to pi #
-> Make array for interpolation #
-> pi : float(%pi);
-> x : pi/10*listn(1,10)?
[0.3141593, 0.6283185, 0.9424778, 1.2566371, 1.5707963, 1.8849556,
2.1991149, 2.5132741, 2.8274334, 3.1415927]
-> y : map(sin,x)?
[0.3090170, 0.5877853, 0.8090170, 0.9510565, 1.0000000, 0.9510565,
0.8090170, 0.5877853, 0.3090170, 0.0000000]
-> Calculate 2nd derivative with spline #
-> yp1 : cos(first(x))?
0.9510565
-> yp10 : cos(last(x))?
-1.0000000
-> Should get -1*y (above) #
-> y2 : spline(x,y,yp1,yp10)?
[-0.3121410, -0.5924815, -0.8157337, -0.9588933, -1.0082536, -0.9589064,
-0.8156812, -0.5926786, -0.3114051, -0.0006038]
```

splint

Description:

Given lists x and y of numbers of length n representing a tabulated function $f(x_i) = y_i$ such that the values of x are strictly increasing, a list $y2a$ as returned by the function **spline**, and a point x, this function returns the value of a cubic spline interpolant at the point x.

splint(xa, ya, y2a, x) $\longrightarrow$ y

Where:

xa is a real list representing the independent variables,
ya is a real list representing the dependent variables,
y2a is a real list as output from the function **spline**,
x is a real number being the point at which the interpolant is required,
y is a real number representing the interpolated value at x.

Example:

In this example the `sin` function is interpolated on $[0, \pi]$ using `spline` and then evaluated using `splint`.

```
-> sine case: sin(x) as 0 < x < pi #
-> mult(x,y) := x*y;
-> xa : map(mult,const(pi,10),
                 [0.05,0.15,0.25,0.35,0.45,0.55,0.65,0.75,0.85,0.95])?
[0.1570796, 0.4712389, 0.7853982, 1.0995574, 1.4137167, 1.7278760, 2.0420351,
2.3561945, 2.6703538, 2.9845130]
-> ya : map(sin,xa)?
[0.1564345, 0.4539905, 0.7071068, 0.8910065, 0.9876883, 0.9876883, 0.8910066,
0.7071068, 0.4539904, 0.1564345]
-> yp1 : cos(first(xa))?
0.9876883
-> yp10 : cos(last(xa))?
-0.9876883
-> y2 : spline(xa,ya,yp1,yp10)?
[-0.1583216, -0.4575768, -0.7129842, -0.8983474, -0.9958405, -0.9958404,
-0.8983474, -0.7129842, -0.4575767, -0.1583216]
-> splint(xa,ya,y2,0.05*pi)?
0.1564345
-> Compare with: #
-> sin(0.05*pi)?
0.1564345
-> splint(xa,ya,y2,0.5*pi)?
0.9999740
-> Compare with: #
-> sin(0.5*pi)?
1.0000000
```

locate

Description:

Given a list xx of numbers which are either monotonically increasing or monotonically decreasing and a number x this function returns an integer j such that x is between x_j and x_{j+1}. If x is less than x_1 then 0 is returned and if x is greater than x_n then n is returned.

locate(xx, x) $\longrightarrow$ j

Where:

xx is a real list which is monotonic,
x is a real number,
j is an integer representing the location of the number x.

Example:

Two points are located, the first using an increasing list and the second a decreasing list.

```
-> locate([1,2,3,4,5,6,7,8,9,10],5.6)?
5
```

---

hunt

Description:

Given a list xx of numbers which are either monotonically increasing or monotonically decreasing, a number x and an initial guess for the position of x in the list, this function returns an integer j such that x is between x_j and x_{j+1}. If x is less than x_1 then 0 is returned and if x is greater than x_n then n is returned.

hunt(xx, x, jlo) $\longrightarrow$ jlo

Where:

xx is a real list which is monotonic,
x is a real number,
jlo is an integer on input being a guess for the location of the number x and on output being the computed location.

Example:

A point is located from within an increasing list of numbers.

```
-> hunt([1,2,3,4,5,6],2.456,1)?
2
```

---

polcoe

Description:

Given lists x and y of length n this function returns an array of polynomial coefficients, also of length, n such that $y_i = \sum cof_j x_i^{j-1}$.

polcoe(x, y) $\longrightarrow$ cof

Where:

x is a real list representing the independent variables,
y is a real list representing the dependent variables,
cof is a real list representing the polynomial coefficients.

Example:

In this example a list of the polynomial coefficients for an interpolant to the exponential function on [0,1] is computed. The values of the interpolant and `exp` are compared at the point 0.2.

```
-> xa : [0.2,0.4,0.6,0.8,1.0];
-> ya : map(exp,xa)?
[1.2214028, 1.4918247, 1.8221189, 2.2255409, 2.7182817]
-> Coefficients #
-> co : polcoe(xa,ya)?
[1.0005323, 0.9940058, 0.5240179, 0.1232891, 0.0764367]
-> exp(0.2)?
1.2214028
-> Compare with: #
-> proj(co,1) + proj(co,2)*(0.2) + proj(co,3)*(0.2)^2
  + proj(co,4)*(0.2)^3 + proj(co,5)*(0.2)^4?
1.2214028
```

polcof

Description:

Given lists x and y of length n this function returns an array of polynomial coefficients also of length n such that $y_i = \sum cof_j x_i^{j-1}$.

polcof(x, y) $\longrightarrow$ cof

Where:

x is a real list representing the independent variables,
y is a real list representing the dependent variables,
cof is a real list representing the polynomial coefficients.

Example:

In this example the coefficients for a polynomial interpolant to the `sin` function are computed and the results compared at $\frac{\pi}{5}$.

```
-> pi : float(%pi)?
3.1415927
-> xa : [0.2*pi,0.4*pi,0.6*pi,0.8*pi,pi]?
[0.6283185, 1.2566371, 1.8849557, 2.5132742, 3.1415927]
-> ya : map(sin,xa)?
[0.5877853, 0.9510565, 0.9510565, 0.5877852, 0.0000000]
-> Coefficients: #
-> co : polcof(xa,ya)?
[-0.0000000, 0.9853291, 0.0524811, -0.2330799, 0.0370958]
-> sin(0.2*pi)?
0.5877853
-> Compare with: #
-> proj(co,1) + proj(co,2)*(0.2*pi) + proj(co,3)*(0.2*pi)^2
  + proj(co,4)*(0.2*pi)^3 + proj(co,5)*(0.2*pi)^4?
0.5877853
```

polin2

Description:

Given lists $x1a$ of length m and $x2a$ of length n of independent variables, and an m by n matrix of function values ya tabulated at the grid points defined by $x1a$ and $x2a$, and given values $x1$ and $x2$ of the independent variables this function returns an interpolated value y and an accuracy indication dy in the $x1$ direction.

polin2(x1a, x2a, ya, x1, x2) −→ [y, dy]

Where:

x1a is a real list representing the first coordinates of tabulated values,
x2a is a real list representing the second coordinates of tabluated values,
ya is a real matrix representing the tabulated function values,
x1 is a real number being the first coordinate of the interpolated point,
x2 is a real number being the second coordinate of the interpolated point,
y is a real number giving the interpolated value,
dy is a real number giving an error estimate for the interpolation.

Example:

In this example the function $g(x, y) = sin(x)e^y$ is interpolated on a rectangle using 25 points.

```
-> g(x,y) := sin(x)*exp(y);
-> pi : float(%pi);
-> x1a : [0.2*pi,0.4*pi,0.6*pi,0.8*pi,pi]?
[0.6283185, 1.2566371, 1.8849557, 2.5132742, 3.1415927]
-> x2a : [0.2,0.4,0.6,0.8,1.0];
-> z:map(exp,x2a);
-> ya : matrix(sin(0.2*pi)*z,sin(0.4*pi)*z,sin(0.6*pi)*z,
               sin(0.8*pi)*z,sin(pi)*z);
-> x1 : 0.7*pi;
-> x2 : 0.3*pi;
-> polin2(x1a,x2a,ya,x1,x2)?
[2.0772289, -0.0139101]
-> Compare with: #
-> g(x1,x2)?
2.0762067
```

bcucof

Description:

Given lists y, $y1$, $y2$ and $y12$ each of length 4, containing the function, gradients and cross derivatives at the four grid points of a rectangular grid cell (numbered counter clockwise from the lower left), and given $d1$, $d2$ the lengths of the grid cell in the first and second coordinate directions, this function returns the 4 by 4 matrix c which is used internally by function **bcuint** for bicubic interpolation.

bcucof(y, y1, y2, y12, d1, d2) $\longrightarrow$ c

Where:

y is a real list being the function values at the grid points,
y1 is a real list being the function derivatives at the grid points in the first coordinate direction,
y2 is a real list being the function derivatives at the grid points in the second coordinate direction,
y12 is a real list being the function second cross derivatives at the grid points,
d1 is a real number being the length of the grid cell in the first direction,
d2 is a real number being the length of the grid cell in the second direction,
c is a real matrix.

Example:

In this example the c matrix for the function $g(s,t) = ste^{-st}$ is computed.

```
-> g(s,t) := s*t*exp(-s*t);
-> g1(x,y) := y*exp(-x*y)*(1.0-x*y);
-> g2(x,y) := x*exp(-x*y)*(1.0-x*y);
-> g12(x,y) := exp(-x*y)*(-3*x*y + x^2*y^2 + 1);
-> x1 : [0.0,2.0,2.0,0.0];
-> x2 : [0.0,0.0,2.0,2.0];
-> d1 : 2;
-> d2 : 2;
-> y : map(g,[1.0,2.0,3.0,4.0],[1.0,2.0,3.0,4.0]);
-> y1 : map(g1,[1.0,2.0,3.0,4.0],[1.0,2.0,3.0,4.0]);
-> y2 : map(g2,[1.0,2.0,3.0,4.0],[1.0,2.0,3.0,4.0]);
-> y12 : map(g12,[1.0,2.0,3.0,4.0],[1.0,2.0,3.0,4.0]);
-> Coefficients for bicubic interpolation: #
-> c:bcucof(y,y1,y2,y12,d1,d2);
```

bcuint

Description:

This function performs bicubic iterpolation within a grid rectangle. An internal call is made to **bcucof**.

bcuint(y, y1, y2, y12, x1l, x1u, x2l, x2u, x1, x2) $\longrightarrow$ [ansy, ansy1, ansy2]

Where:

y is a real list being the function values at the grid points,
y1 is a real list being the function derivatives at the grid points in the first coordinate direction,
y2 is a real list being the function derivatives at the grid points in the second coordinate direction,
y12 is a real list being the function second cross derivatives at the grid points,
x1l is a real number being the lower coordinate of the grid cell in the first direction,
x1u is a real number being the upper coordinate of the grid cell in the first direction,
x2l is a real number being the lower coordinate of the grid cell in the second direction,
x2u is a real number being the upper coordinate of the grid cell in the second direction,
x1 is a real number being the first coordinate of the interpolation point,
x2 is a real number being the second coordinate of the interpolation point,
ansy is the interpolated function value,
ansy1 is the interpolated derivative in the first direction,
ansy2 is the interpolated derivative in the second direction.

Example:

In the example below bicubic spline interpolation is used for the function $g(x, y) = x^2 y^2$ and the results checked.

```
-> g(x,y) := (x*y)^2;
-> g1(x,y) := 2*x*y^2;
-> g2(x,y) := 2*y*x^2;
-> g12(x,y) := 4*x*y;
-> xx : [0.0,2.0,2.0,0.0];
-> yy : [0.0,0.0,2.0,2.0];
-> x1l : proj(xx,1);
-> x1u : proj(xx,2);
-> x2l : proj(yy,1);
-> x2u : proj(yy,4);
-> y : map(g, xx, yy)?
[0.0, 0.0, 16.0, 0.0]
-> y1 : map(g1, xx, yy)?
[0.0, 0.0, 16.0, 0.0]
-> y2 : map(g2, xx, yy)?
[0.0, 0.0, 16.0, 0.0]
-> x1 : 0.2; x2 : x1;
-> bcuint(y,y1,y2,y12,x1l,x1u,x2l,x2u,x1,x2)?
[0.001600000953674336d0, 0.016000000715255747d0, 0.016000000715255747d0]
-> [g(x1,x2),g1(x1,x2),g2(x1,x2)]?
[0.0016000002, 0.016, 0.016]
-> x1 : 0.4; x2 : x1;
-> bcuint(y,y1,y2,y12,x1l,x1u,x2l,x2u,x1,x2)?
[0.025600000152587894d0, 0.12800000572204598d0, 0.12800000572204598d0]
-> [g(x1,x2),g1(x1,x2),g2(x1,x2)]?
[0.025600003, 0.128, 0.128]
-> x1 : 0.6; x2 : x1;
-> bcuint(y,y1,y2,y12,x1l,x1u,x2l,x2u,x1,x2)?
[0.12960002059936646d0, 0.43200005149841514d0, 0.43200005149841514d0]
-> [g(x1,x2),g1(x1,x2),g2(x1,x2)]?
[0.1296, 0.43200004, 0.43200004]
```

splie2

Description:

Given an m by n tabulated function ya and tabulated independent variables $x1a$ (a list of m values) and $x2a$ (a list of n values), this routine constructs one-dimensional natural cubic splines of the rows of ya and returns the matrix of second derivatives.

splie2(x1a, x2a, ya) $\longrightarrow$ y2a

Where:

x1a is a real list of the tabulated independent variables in the first coordinate direction,
x2a is a real list of the tabulated independent variables in the second coordinate direction,
ya is a real matrix of the tabulated function values,
y2a is a real matrix of the second derivatives in the second direction of the one-dimensional splines.

Pre-variables:

splie2_nn [100] an integer giving a workspace dimension.

Example:

In this example `splie2` is used to compute a table of second derivatives of one dimensional cubic splines approximating the function $g(x, y) = x^2y^2$ on a 5 by 5 grid.

```
-> x1 : [0.2,0.4,0.6,0.8,1.0]; x2 : x1; g(x1,x2) := x1*x1*x2*x2;
-> y : matrix(map(g,const(0.2,5),x2),map(g,const(0.4,5),x2),
map(g,const(0.6,5),x2), map(g,const(0.8,5),x2),map(g,const(1.0,5),x2))?
| 0.0016000  0.0064000  0.0144000  0.0256000  0.0400000 |
|                                                       |
| 0.0064000  0.0256000  0.0576000  0.1024000  0.1600000 |
|                                                       |
| 0.0144000  0.0576000  0.1296000  0.2304000  0.3600000 |
|                                                       |
| 0.0256000  0.1024000  0.2304000  0.4096000  0.6400000 |
|                                                       |
| 0.0400000  0.1600000  0.3600000  0.6400000  1.0000000 |
-> splie2(x1,x2,y)?
| 0.0000000  0.1028572  0.0685714  0.1028572  0.0000000 |
|                                                       |
| 0.0000000  0.4114287  0.2742855  0.4114290  0.0000000 |
|                                                       |
| 0.0000000  0.9257147  0.6171422  0.9257150  0.0000000 |
|                                                       |
| 0.0000000  1.6457146  1.0971421  1.6457159  0.0000000 |
|                                                       |
| 0.0000000  2.5714267  1.7142893  2.5714252  0.0000000 |
```

splin2

Description:

Given an m by n tabulated function ya and tabulated independent variables $x1a$ (a list of m values) and $x2a$ (a list of n values), the matrix of second derivatives $y2a$ as output by $splie2$ and the coordinates $x1$, $x2$ of a desired interpolating point, this routine computes an interpolated value y by bicubic spline interpolation.

splin2(x1a, x2a, ya, y2a, x1, x2) $\longrightarrow$ y

Where:

x1a is a real list of the tabulated independent variables in the first coordinate direction,
x2a is a real list of the tabulated independent variables in the second coordinate direction,
ya is a real matrix of the tabulated function values,
y2a is a real matrix of the second derivatives of the one-dimensional splines as output by **splie2**,
x1 is a real number being the first coordinate of the interpolation point,
x2 is a real number being the second coordinate of the interpolation point,
y is a real number being the interpolated value.

Example:

In this example `splie2` and `splin2` are used to interpolate the function $g(x, y) = xye^{-xy}$ on a 5 by 5 grid over the unit square.

```
-> x1 : [0.2,0.4,0.6,0.8,1.0]; x2 : x1;
-> g(x1,x2) := x1*x2*exp(-x1*x2);
-> y : matrix(map(g,const(0.2,5),x2), map(g,const(0.4,5),x2),
              map(g,const(0.6,5),x2), map(g,const(0.8,5),x2),
              map(g,const(1.0,5),x2))?
| 0.0384316  0.0738493  0.1064305  0.1363430  0.1637462 |
|                                                       |
| 0.0738493  0.1363430  0.1887907  0.2323677  0.2681280 |
|                                                       |
| 0.1064305  0.1887907  0.2511635  0.2970160  0.3292870 |
|                                                       |
| 0.1363430  0.2323677  0.2970160  0.3374671  0.3594632 |
|                                                       |
| 0.1637462  0.2681280  0.3292870  0.3594632  0.3678795 |
-> y2 : splie2(x1,x2,y);
-> splin2(x1,x2,y,y2,0.4,0.8)?
0.2323677
-> Compare with: #
-> g(0.4,0.8)?
0.2323677
```