

Math 128a, Section 3 — Solution Set 2 — Due Sept 13, 2001

(1) Problem 2.1.1. Test with $n = 20$ random values y_j interpolated at $x_j = 10^6 : 10^6 + 10^3 j$. Use (a) $u = 0, v = 1$ and (b) $u = 10^6, v = 10^3 n$. Check whether your polynomials match the values y_j at the interpolation points x_j : report the maximum errors in each case.

(Solution 1) Only a slight modification of the book's script `InterpV` is required: shift and scale the variable x . Thus we have

```
function a = InterpV(x,y,u,v)
% a = InterpV(x,y,u,v)
% This computes the Vandermonde polynomial interpolant where
% x is a column n-vector with distinct components and y is a
% column n-vector.
%
% a is a column n-vector with the property that if
%
%       p(x) = a(1) + a(2)((x-u)/v) + ... a(n)((x-u)/v)^(n-1)
% then
%       p(x(i)) = y(i), i=1:n
% If v is not specified then it is set to 1,
% while if u is not specified then it is set to 0.

if( nargin <= 3 )
    v = 1;
end

if( nargin <= 2 )
    u = 0;
end

n = length(x);
V = ones(n,n);
for j=2:n
    % Set up column j.
    V(:,j) = ((x-u)./v).*V(:,j-1);
end
a = V\y;
```

We run this program with the following function p211(n):

```
function [ ea, eb ] = p211( n )
clc;
y = rand( n, 1 )
x = [10^6+10^3 : 10^3 : 10^6+n*10^3]';
format short e;
ca = InterpVuv( x, y )
ya = HornerV( ca, x )
ea = max( abs( ya - y ) );
u = 10^6;
v = 10^3 * n;
cb = InterpVuv( x, y, u, v )
yb = HornerV( cb, (x-u)./v )
eb = max( abs( yb - y ) );
```

The results for $n = 3$ and 20 show the superiority of shifting and scaling to the appropriate interval:

```
[ea,eb] = p211(3)
```

```
y = 0.9501 0.2311 0.6068
```

```
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 3.306794e-19.
```

```
ca = 5.4971e+05 -1.0971e+00 5.4735e-07
```

```
ya = 9.5013e-01 2.3114e-01 6.0684e-01
```

```
cb = 2.7638e+00 -7.0831e+00 4.9261e+00
```

```
yb = 9.5013e-01 2.3114e-01 6.0684e-01
```

```
ea = 4.2342e-11
```

```
eb = 3.3307e-16
```

```
[ea,eb] = p211(20)
```

y = 4.5647e-01 1.8504e-02 8.2141e-01 4.4470e-01 6.1543e-01
7.9194e-01 9.2181e-01 7.3821e-01 1.7627e-01 4.0571e-01
9.3547e-01 9.1690e-01 4.1027e-01 8.9365e-01 5.7891e-02
3.5287e-01 8.1317e-01 9.8613e-03 1.3889e-01 2.0277e-01

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 2.531749e-132.

ca = 2.8589e+15 -1.6214e+10 3.4337e+04 -3.1245e-02 8.6256e-09
2.4859e-15 -3.6274e-21 2.8711e-27 1.0697e-32 -2.0777e-38
1.4411e-44 -6.1949e-51 1.4442e-57 -2.2762e-63 4.5274e-69
4.2415e-76 -2.4709e-81 -2.2747e-87 3.3865e-93 -9.8816e-100

ya = -4.5000e+00 -3.0000e+00 -6.5000e+00 -1.1000e+01 -3.0000e+00
-4.0000e+00 -2.0000e+00 -1.0000e+00 0 -2.0000e+00
0 -4.5000e+00 -3.0000e+00 -4.5000e+00 -1.0000e+00
-1.0000e+00 -4.0000e+00 -5.0000e+00 -9.5000e+00 -3.0000e+00

Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 4.796981e-18.

cb = -6.1865e+04 4.3441e+06 -1.3285e+08 2.3871e+09 -2.8545e+10
2.4321e+11 -1.5402e+12 7.4535e+12 -2.8078e+13 8.3328e+13
-1.9616e+14 3.6714e+14 -5.4517e+14 6.3778e+14 -5.8014e+14
4.0152e+14 -2.0414e+14 7.1842e+13 -1.5625e+13 1.5815e+12

yb = 4.5647e-01 1.4980e-02 8.3186e-01 4.3045e-01 6.7830e-01
8.4010e-01 9.8441e-01 8.4220e-01 2.7147e-01 5.1576e-01
1.0785e+00 1.1018e+00 4.4977e-01 9.7353e-01 1.5713e-01
4.7419e-01 1.0252e+00 1.3334e-01 3.0346e-01 3.4011e-01

ea = 1.1445e+01

eb = 2.1202e-01

I have compacted the display of vectors from rows to columns to save paper.

(2) Problem 2.2.1. Also write a script `V2N` which converts polynomials, represented as vectors of coefficients, from the Vandermonde (power basis) representation to the Newton representation. Test them by converting the two polynomials from problem 1 from Vandermonde to Newton and back again: report the maximum error in the coefficients.

(Solution 2) I can convert from any representation to any other representation of the degree- $(n-1)$ polynomial $p(x)$ which interpolates n values y_j at n points x_j by evaluating p at any desired n points z_j , and building the desired representation of p from the values $p(z_j)$. For Newton to Vandermonde, the natural evaluation points are the n points x specified in the Newton representation. However, we might not know x_n , so we use instead equidistant points on the interval where the x_i 's live—even though equidistant polynomial interpolation may not be very accurate near the ends of the interval. Thus `N2V` might read

```
function a = N2V( c, x )
% function a = N2V( c, x )
% p(z) = c(1) + c(2)(z-x(1)) + ... + c(n)(z-x(1))...(z-x(n-1))
%      = a(1) + a(2) * z + ... + a(n) * z^(n-1)
%
z = linspace( min( x ), max( x ), length( c ) )
% Evaluate p(z) at the n points z(1),...,z(n) and construct the
% Vandermonde interpolant of (z(i),p(z(i))), i=1:n
a = InterpV( z, HornerN( c, x, z ) );
```

We can reverse the process very similarly: `V2N` might read

```
function c = V2N( a, x )
% function c = V2N( a, x )
% converts Vandermonde coefficients a(1:n) to Newton coeffs c(1:n)
% based on the points x(1:n).
%      p(z) = a(1) + a(2) * z + ... + a(n) * z^(n-1)
%      = c(1) + c(2)(z-x(1)) + ... + c(n)(z-x(1))...(z-x(n-1))
c = InterpN( x, HornerV( a, x ) );
```

To test these conversions on the polynomials of the previous problem, we use a script `p221(n)` which also tests the Vandermonde conversion based on

shifted and scaled points z and reports the resulting floating-point error in ec :

```
function [ ea, eb, ec ] = p221( n )
clc;
y = rand( n, 1 );
x = [10^6+10^3 : 10^3 : 10^6+n*10^3]';
format short e;
ca = InterpVuv( x, y )
u = 10^6;
v = 10^3 * n;
cb = InterpVuv( x, y, u, v )

ea = max( abs( N2V( V2N( ca, x ), x ) - ca ) );
eb = max( abs( N2V( V2N( cb, x ), x ) - cb ) );
ec = max( abs( N2V( V2N( cb, (x-u)./v ), (x-u)./v ) - cb ) );
```

The results show the superiority of shifting and scaling:

```
[ea,eb,ec] = p221(3)
Warning: Matrix is close to singular or badly scaled.
          Results may be inaccurate. RCOND = 3.306794e-19.
ca = -1.6221e+05  3.2368e-01  -1.6147e-07

cb = -1.2775e-01  2.2023e+00  -1.4532e+00

Warning: Matrix is close to singular or badly scaled.
          Results may be inaccurate. RCOND = 3.306794e-19.
Warning: Matrix is close to singular or badly scaled.
          Results may be inaccurate. RCOND = 3.306794e-19.

ea = 2.4813e-06

eb = 2.9299e+02

ec = 2.2204e-16
```

```
[ea,eb,ec] = p221(20)
Warning: Matrix is close to singular or badly scaled.
```

Results may be inaccurate. RCOND = 2.531749e-132.

```
ca = -1.0327e+16 4.5650e+10 -6.6951e+04 2.5188e-02 1.5198e-08
     -3.1683e-15 4.6543e-21 -1.8278e-26 -6.5008e-33 3.5239e-38
     -3.1460e-44 5.4976e-51 9.5201e-57 -1.1963e-63 4.8275e-70
     -7.8619e-75 2.1187e-81 4.5025e-87 -2.6250e-93 3.1723e-100
```

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = 4.796981e-18.

```
cb = 5.8396e+04 -4.1430e+06 1.2823e+08 -2.3343e+09 2.8291e+10
     -2.4429e+11 1.5674e+12 -7.6800e+12 2.9272e+13 -8.7826e+13
     2.0883e+14 -3.9445e+14 5.9061e+14 -6.9610e+14 6.3742e+14
     -4.4378e+14 2.2681e+14 -8.0185e+13 1.7509e+13 -1.7783e+12
```

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = 2.531749e-132.

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = 2.531749e-132.

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = 4.539442e-18.

```
ea = 7.3892e+16
```

```
eb = 2.8565e+127
```

```
ec = 2.1197e+14
```

(3) Problem 2.4.4. Test your script on $f(x, y) = \cos(10x) \sin(2y)$ on the square $0 \leq x \leq 1$, $0 \leq y \leq 1$. Use $n = 4, 8, 16, 32, 64$ and $x = 0.1$, $y = 0.2$. Note that (x, y) is in an edge tile on the first few meshes. Prove an error bound analogous to Theorem 2 of the text for cubic interpolation in two dimensions and check that your tests agree with the error bound.

(Solution 3) The following scripts (modified from Van Loan's web site) do two-dimensional cubic interpolation of $f(x, y) = \cos(10x) \sin(2y)$:

```
e = p244(4,4)
```

```
e = 1.3789e-01
```

```

for i = [ 8 16 32 64 128 ]
e = [ e p244(i,i)];
end
e = 1.3789e-01    8.0373e-03    6.1151e-07    1.4911e-05    1.1422e-06    7.2227e-08

```

Observe that the error decreases like $O(n^{-4})$ as $n \rightarrow \infty$: Each time n doubles, the error decreases by roughly 16.

```

function e = p244( n, m )
a = 0; b = 1;
c = 0; d = 1;
fA = SetUp('f',a,b,n,c,d,m);
xc = 0.2;
yc = 0.1;
exact = f(xc,yc);
approx = CubicInterp2D(xc,yc,a,b,n,c,d,m,fA);
e = abs(approx-exact);

```

```

function z = CubicInterp2D(x,y,a,b,n,c,d,m,fA)
%
% n,m:      integers >= 2
% x,a,b:   scalars that satisfy a<x<b
% y,c,d:   scalars that satisfy c<y<d
% fA:      an n-by-m matrix with the property that
%           fA(i,j) = f(a+(i-1)(b-a)/(n-1),c+(j-1)(d-c)/(m-1))
%           for some function f(.,.).
% Post:
%   z:      cubically interpolated value for f(x,y).
%
% Determine where in the grid the point is located.

hx = (b-a)/(n-1);
i = max([1 ceil((x-a)/hx)]);
hy = (d-c)/(m-1);
j = max([1 ceil((y-c)/hy)]);
% Adjust for edge effects
if (1 < i)&(i < n-1)
    % not on an edge

```

```

    ival = (i-1):(i+2);
elseif 1==i
    % on left edge
    ival = 1:4;
else
    % on right edge
    ival = n-3:n;
end
if (1 < j) & (j < m-1)
    % not on an edge
    jval = (j-1):(j+2);
elseif 1==j
    % on bottom edge
    jval = 1:4;
else
    % on top edge
    jval = m-3:m;
end
% Determine coordinates of the grid points.
xval = a + (ival-1)*hx;
yval = c + (jval-1)*hy;
% call Small2D with the appropriate 4x4 subgrid:

z = Small2D(x,y,xval',yval',fA(ival,jval));

function z = Small2D(x,y,xval,yval,fval)
%
% x and y are scalars representing the point at which to interpolate f
% xval and yval are column 4-vectors with
%     xval(1) < xval(2) < xval(3) < xval(4)
%     yval(1) < yval(2) < yval(3) < yval(4)
% fval is a 4x4 matrix of evaluations of f at the xval,yval coords.
%
% Let p1 be the cubic interpolant of (xval(q),fval(q,1)), q=1:4.
% Let p2 be the cubic interpolant of (xval(q),fval(q,2)), q=1:4.
% Let p3 be the cubic interpolant of (xval(q),fval(q,3)), q=1:4.
% Let p4 be the cubic interpolant of (xval(q),fval(q,4)), q=1:4.
%
```

```

% Let p be the cubic interpolant of the four points
%           (yvals(1),p1(x))
%           (yvals(2),p2(x))
%           (yvals(3),p3(x))
%           (yvals(4),p4(x))
%
%       z = p(y).

fy = zeros(4,1);
for k=1:4
    c = InterpN(xvals,fvals(:,k)); % interpolate horizontally
    fy(k) = HornerN(c,xvals,x);    % evaluate interpolants at x
end

c = InterpN(yvals,fy); % interpolate vertically
z = HornerN(c,yvals,y); % evaluate interpolant at y

```

To conclude, we prove an error bound. Suppose without loss of generality that we are interpolating values of a smooth function $f(x, y)$ from 16 points (ih, jh) spaced a distance h apart to a point (x, y) . Let $\varphi_i(x)$ be the i th Lagrange basis function. Then horizontal interpolation to each of the four points (x, jh) produces a value

$$q(x, jh) = \sum_{i=1}^4 f(ih, jh)\varphi_i(x) = f(x, jh) + e_j(x)$$

where the error in each horizontal interpolation is given by

$$e_j(x) = \frac{\partial^4 f}{\partial y^4}(\zeta(jh), jh) \frac{1}{4!}(x - 0h)(x - 1h)(x - 2h)(x - 3h)$$

for some unknown function $\zeta(y)$. Interpolating in the y variable then produces a value

$$\begin{aligned}
 p(x, y) &= \sum_{j=1}^4 q(x, jh)\varphi_j(y) \\
 &= \sum_{j=1}^4 f(x, jh)\varphi_j(y) + \sum_{j=1}^4 e_j(x)\varphi_j(y) \\
 &= f(x, y) + E(x, y) + \sum_{j=1}^4 e_j(x)\varphi_j(y)
 \end{aligned}$$

where

$$E(x, y) = \frac{\partial^4 f}{\partial x^4}(x, \eta(x)) \frac{1}{4!} (y - 0h)(y - 1h)(y - 2h)(y - 3h)$$

for some function $\eta(x)$. Thus if all fourth derivatives of f are bounded by a constant M_4 , then the total error in two-dimensional cubic interpolation is bounded by

$$|E(x, y) + \sum_{j=1}^4 e_j(x) \varphi_j(y)| \leq \frac{M_4}{4!} 5Ch^4$$

where the Lagrange basis functions and the polynomial $\omega(x) = x(x-1)(x-2)(x-3)$ are bounded by some constant C .

For our test function $f(x, y) = \cos(10x) \sin(2y)$ we have $M_4 = 10^4$, so with $h = 1/64$, for example, the error should be less than

$$(5/24)10^4 C 64^{-4} = 10^{-4} C$$

which does not contradict with the experimental result of about 10^{-6} if the constant C is of order unity.

(4) Problem 2.4.7. Prove that $P^T P$ is diagonal for general n . (Hint: use $\cos(x) = (e^{ix} + e^{-ix})/2$ and a finite geometric series.) Verify the flop count of your modified script is $O(n^2)$ for $n = 4, 8, \dots, 128$.

(Solution 4) We need to show that each column P_j of P is orthogonal to any other row P_k in the sense that $P_j^T P_k = 0$. Take a column of cosines for example:

$$\begin{aligned} P_j^T P_k &= \sum_{l=0}^{n-1} \cos(jl\pi/m) \cos(kl\pi/m) \\ &= \frac{1}{4} \sum_{l=0}^{n-1} (e^{ijl\pi/m} + e^{-ijl\pi/m})(e^{ikl\pi/m} + e^{-ikl\pi/m}) \\ &= \frac{1}{4} \sum_{l=0}^{n-1} e^{i(j+k)l\pi/m} + e^{-i(j+k)l\pi/m} + e^{i(j-k)l\pi/m} + e^{-i(j-k)l\pi/m} \\ &= \frac{1}{4} \left(\frac{1 - e^{i(j+k)2m\pi/m}}{1 - e^{i(j+k)\pi/m}} + \frac{1 - e^{-i(j+k)2m\pi/m}}{1 - e^{-i(j+k)\pi/m}} + \frac{1 - e^{i(j-k)2m\pi/m}}{1 - e^{i(j-k)\pi/m}} + \frac{1 - e^{-i(j-k)2m\pi/m}}{1 - e^{-i(j-k)\pi/m}} \right) \\ &= 0 \end{aligned}$$

since $e^{2\pi i} = 1$. This calculation fails if $j = k$ or $j + k = n = 2m$, but that can't happen since $j \neq k$ and both j and k lie between 0 and m . Hence $P^T P$ is diagonal. Therefore we can replace the $O(n^3)$ operation of inverting P by the $O(n^2)$ operation of multiplying by P^T and the $O(n)$ operation of inverting the diagonal matrix $P^T P$:

```

y = P' * f;
for i = 1:n
    y(i) = y(i) / (P(:,i))' * P(:,i))
end

```

For part(b), we observe that there are only $O(n)$ distinct sines and cosines at spacings $2\pi/n$, so we can certainly tabulate them once and look them up when needed. Thus the number of trigonometric evaluations needed is only $O(n)$.