Numerical Analysis
MS-C1650
Tölle/Nyman
Exercises (MATLAB), Week 18, 2022

DATE[1]

On assignments: Submit homework to your assistant electronically via the course pages. MATLAB-assignments are submitted via Peergrade.

1 Univariate Barycentric Formulation

We have seen that Newton's interpolation polynomials appear to have an advantage in that they can be updated more efficiently than for instance Lagrange's interpolation polynomial. However, the Lagrange form can be written more efficiently in the so-called barycentric form, where the evaluation is faster.
Let us introduce the following quantities

$$\varphi(x) = \prod_{j=0}^{n}(x - x_j) \text{ and } w_i = \frac{1}{\prod_{j\neq i}(x_i - x_j)}.$$

EXERCISE 2

(a) Write a function `lagweights.m` that computes the weights w_k for the nodes x_k.

(b) Write a function `specialsum.m` that computes the quantity $\sum_{i=0}^{n} \frac{z_i}{t-x_i}$, when x and z are arrays of size n and t is an array of size s. The output has to be an array of size s. That is, t has the values where the interpolation polynomial is evaluated.

(c) Write a program `lagpolint.m` that computes the barycentric form of p at points t.

[1] Published on 2022-04-28

(d) Test lagpolint.m by sampling from the function $y = \sqrt{|t|}$ on $[-1, 1]$. Try first 9 uniform points and then 101 Chebyshev points

$$x_j = -\cos(j\pi/n), \quad j = 0, 1, \ldots, 100 := n.$$

Plot the polynomials.

lagweights.m is just the formula for the weights.

```
>> W =lagweights ( [ 1, 2, 3 ]' )

W =

    0.5000
   -1.0000
    0.5000
```

specialsum.m is perhaps the best possible name for this routine. However, it can be used very flexibly in the sequel. If one of the ts is one of the xs, nothing bad happens, we just get NaN.

```
>> S =specialsum ((0:3)',(1.5:0.5:3)', -4:0)

S =

   -1.6202   -2.0000   -2.6417   -4.0833       NaN
```

lagpolint.m builds on the two previous routines, and could look something like this:

```
Y=sqrt(abs(X));
W=lagweights(X);
T=1:0.002:1;
S1=specialsum(...);
S2=specialsum(...);
P=S1./S2;
plot(X,Y,'.b',T,P,'r')
```

2 Application to Interpolating Surfaces

Let us next consider a regular grid of points (x_i, y_j) and the surface values $z_{ij} = f(x_i, y_j)$. Let $\mathbf{x} = (x_0, \ldots, x_m) \in \mathbb{R}^{m+1}$ and $\mathbf{y} = (y_0, \ldots, y_n) \in$

$\mathbb{R}^{n+1}$. In this setting the surface can be interpolated using a product (tensor product) of univariate interpolation polynomials. In the sequel we denote the y-dependent quantities with a bar, for instance, $\bar{l}_q(t)$ for the corresponding Lagrange basis polynomial in the y-direction.

EXERCISE 4 Write a program interpolsurf.m such that given the grid points as x and y, and the sampled values z $= (f(x_i, y_j)) \in \mathbb{R}^{(m+1)\times(n+1)}$, computes the values of the polynomial $P(s,t)$.
Test with $f(s,t) = \sin(s+t)$, $m = 3$, x a uniform partition of $[0,\pi]$, and $n = 7$, y a uniform partition of $[0,2\pi]$. Plot $P(s,t)$, $f(s,t)$, and the difference $f - P$.

interpolsurf.m is more involved. However, after some breathing and reflection, one realises that we have done the heavy lifting above. It's just a case of fitting the pieces together. Here is one outline, with f being the surface function.

```
m = 3;
n = 7;

X=(0:1/m:1)' * pi;
Y=(0:1/n:1)' * 2 * pi;
[YY,XX ]=meshgrid(Y,X);
ZZ =f (XX,YY);
U= ( 0: 0.02: 1 )' * pi;
V= ( 0: 0.02: 1 )' * 2 * pi;
W1 =...;
W2 =...;
% numerator
for p=1:length(X)
 S1(p,:)= specialsum (...);
end
for j =1:length(V)
 Suv (:,j)=specialsum (...)'; % Lucky that S1 is available!
end
% denominator
SX =specialsum (...);
SY =specialsum (...);
[SSX,SSY]=meshgrid(SX,SY);
Slag= Suv./(SSX .* SSY)';
subplot(121)
[VV,UU ]=meshgrid(V,U);
Sf=f (UU,VV);
surf(UU,VV,Sf )
title( 'The  function ' )
```