```
function [e1,e2]=compare_poly_spline(m,n,a)

  %i)

  f=@(t)exp(t-5);

  %ii)

  x=linspace(4,6,m);

  %iii) add noise values

  y=f(x);

  noise=a*(rand(1,m)-0.5);

  for i=1:m

                y(1,i)=y(1,i)+ noise(1,i);

  end

  %iv)least squares polynomial approximation

  V=zeros(m,n+1);

  full=vander(x);

  for i=1:m

     for j=1:n+1

        V(i,j)=full(i,j);

     end

  end

  coeff=(V'*V)\(V'*y');



  %v)least squares spline approximation

  % n subintervals<m

  spline=spap2(n,4,x,y);

  %vi)compute the error

  err1=f(x)-polyval(coeff,x);

  err2=f(x)-spval(spline,x);
```

```
   e1=norm(err1,2);

   e2=norm(err2,2);

end
```

for function f=@(t)exp(t-5);

```
%increasing noise level a

for i=0:10:100

        [e1,e2]=compare_poly_spline(10,7,i)

end
```

e1 =4.365880120042917

e2 =1.212381843132080e-15

e1 =4.527821104717662

e2 =9.571496837133818

e1 =4.279426523533919

e2 = 21.307247613711077

e1 =4.571141317230891

e2 =26.414523523006110

e1 =4.577635211321507

e2 =39.751945653879325

e1 =4.916671239611137

e2 =30.999360849874918

e1 =5.223837851287065

e2 =51.989377033065630

e1 =5.936588262962719

e2 =68.685217752775586

e1 =6.257666438215535

e2 =67.631407278928336

e1 =4.788109853256718

e2 =69.855139146371116

e1 =5.893727291206094

e2 =78.772246457530798

least squares polynomial approximation’s error grows very slowly as the noise level grows.

least squares spline approximation’s error grows significantly as the noise level grows.

So the former method is more suitable for points that have high noise levels.

```
%increasing number of points m

for i=10:10:100

        [e1,e2]=compare_poly_spline(i,round(i*0.8),3)

end
```

e1 =3.791398244225777

e2 =2.754248774288173

e1 =6.164260122139730

e2 = 3.865800259392176

e1 =7.510272047780560

e2 =4.253789674922318

e1 =8.634042464179917

e2 =4.379966527253830

e1 =9.626503433803473

e2 =5.785629131612531

e1 =10.525946647642526

e2 = 6.222183048179527

e1 =11.354466951945506

e2 =6.235205069822720

e1 =12.126573584496596

e2 =6.371297556106724

e1 =12.852418076728007

e2 =7.286765032316611

e1 =13.539431617221776

e2 =7.773194815039095

As m increases the error grows for both methods. So increasing m does not really help decrease the error.

%As the number of subintervals/equations n increases

```
for i=10:10:100

        [e1,e2]=compare_poly_spline(110,i,3)

end
```

e1 =14.193247339962106

e2 = 2.863625908391694

e1 =14.193247339962106

e2 =4.799792018162312

e1 =14.193247339962106

e2 =4.085296846167828

e1 =14.193247339962106

e2 =5.572664160984115

e1 =14.193247339962106

e2 =6.409580736960801

e1 = 14.193247339962106

e2 =6.697816269366837

e1 =14.193247339962106

e2 =7.338025584835441

e1 =14.193247339962106

e2 = 7.730666444523881

e1 =14.193247339961180

e2 =8.732684259149844

e1 =14.193244063261981

e2 =9.030589888941815

In this case, as n grows the error for the least square spline approximation grows but the least square polynomial approximation error almost stays the same.

Trying the rest of the functions

```
g=@(t) sin(t);

%increasing noise level a

for i=0:30:90

        [e1,e2]=compare_poly_spline(10,7,i)

end
```

e1 =2.437828518258076

e2 =5.578801654593729e-16

e1 =2.906761004372050

e2 = 25.739205230040930

e1 = 3.353429590920431

e2 =41.660974046233392

e1 =3.973693688900310

e2 =73.926863679077670

```
%increasing number of points m

 for i=10:30:100

        [e1,e2]=compare_poly_spline(i,round(i*0.8),3)

end
```

```
e1 =1.959728911466477

e2 =2.843717650509940

e1 =5.220792949297352

e2 =5.484866323513537

e1 =6.926967700574505

e2 = 6.358904113930707

e1 =8.288922770985042

e2 =8.001048706510673


h=@(x) log(x-3);

%increasing noise level a

for i=0:30:90

        [e1,e2]=compare_poly_spline(10,7,i)

end

e1 =  2.213273841658525

e2 = 4.743999203137125e-16

e1 =2.700734389020463

e2 =30.270620915208262

e1 = 3.832001816115155

e2 = 59.848687067927266

e1 =4.288410202573069

e2 = 80.125088289247913


%increasing number of points m

for i=10:30:100

        [e1,e2]=compare_poly_spline(i,round(i*0.8),3)

end
```

e1 =1.838885600429851

e2 =2.394407011540068

e1 =4.546996085073390

e2 = 5.643332122754041

e1 =6.009302539794117

e2 =7.027277423365595

e1 =7.179747792858250

e2 =7.746331679935520

```
k=@(x)abs(x-5).^(3/2);

%increasing noise level a

for i=0:30:90

        [e1,e2]=compare_poly_spline(10,7,i)

end
```

e1 =1.752680717369853

e2 =2.844946500601964e-16

e1 = 1.553992457217242

e2 = 23.518895336487379

e1 = 2.561814461002111

e2 =61.750495362977126

e1 =3.177844578845265

e2 =78.178466414016114

```
%increasing number of points n

for i=10:30:100

        [e1,e2]=compare_poly_spline(i,round(i*0.8),3)

end
```

e1 = 1.454737395571999

e2 =2.625277254193176

e1 =3.282528473393537

e2 =5.121154779440801

e1 =4.273697678515359

e2 =6.401461097568831

e1 = 5.075440941672766

e2 =8.346804672265510

For all 4functions the effect of growing a and m are pretty much the same. By increasing 'a' (noise level) the error for the least square polynomial approximation grows very slowly, and the error for the least square spline approximation increases much faster than the previous method.

For both method, as m(number of points) increases both error increases. I would say that K.P. Kaypee and T.N. Tian. Nigel is wrong. Trish is right.