MATH661 Homework 3 - Least squares problems

Posted: 09/09/21

Due: 09/15/21, 11:55PM

This assigment addresses one of the fundamental topics within scientific computation: finding economical descriptions of complex objects. Some object is described by $\boldsymbol{y} \in \mathbb{C}^m$ (with m typically large), and a reduced description is sought by linear combination $\boldsymbol{A}\boldsymbol{x}$, with $\boldsymbol{A} \in \mathbb{C}^{m \times n}$ ($n < m$, often $n \ll m$). The surprisingly simple Euclidean geometry of Fig. 1 (which should be committed to memory) will be shown to have wide-ranging applicability to many different types of problems. The error (or residual) in approximating $\boldsymbol{y}$ by $\boldsymbol{A}\boldsymbol{x}$ is defined as

$$\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x},$$

and 2-norm minimization defines the least-squares problem

$$\min_{\boldsymbol{x} \in \mathbb{C}^m} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|.$$

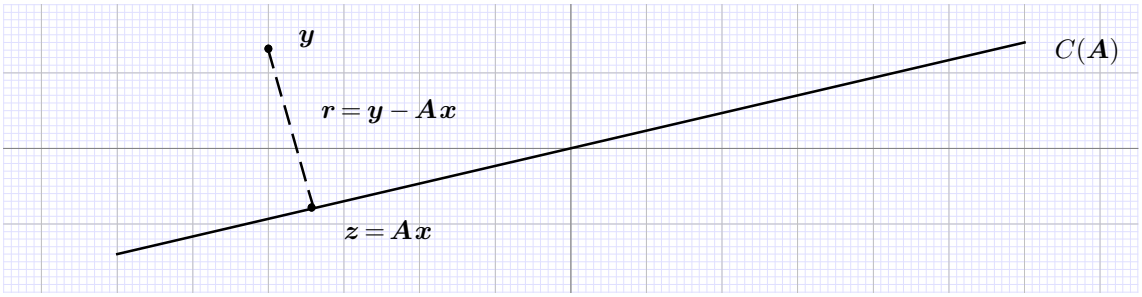

Figure 1. Least squares (2-norm error minimization) problem.

1 Track 1

Consider data $\mathcal{D} = \{(t_i, y_i) | i = 1, 2, ..., m\}$ obtained by sampling a function $f: \mathbb{R} \to \mathbb{R}$, with $y_i = f(t_i)$. An approximation is sought by linear combination

$$f(t) \cong x_1\, a_1(t) + x_2\, a_2(t) + \cdots + x_n\, a_n(t).$$

Introduce the vector-valued function $A: \mathbb{R} \to \mathbb{R}^n$ (organized as a row vector)

$$A(t) = [\ a_1(t)\ \ a_2(t)\ \ \ldots\ \ a_n(t)\ ],$$

such that

$$f(t) \cong A(t)\,\boldsymbol{x}, \boldsymbol{x} = [\ x_1\ \ x_2\ \ \ldots\ \ x_n\ ]^T.$$

With $\boldsymbol{t} = [\ t_1\ \ t_2\ \ \ldots\ \ t_m\ ]^T$ a sampling of the function domain, a matrix is defined by

$$\boldsymbol{A} = A(\boldsymbol{t})\,\boldsymbol{x} = [\ a_1(\boldsymbol{t})\ \ a_2(\boldsymbol{t})\ \ \ldots\ \ a_n(\boldsymbol{t})\ ]\,\boldsymbol{x} \in \mathbb{R}^{m \times n}.$$

Tasks. In each exercise below, construct the least-squares approximant for the stated range of $n \in \mathcal{N}$, sample points $\boldsymbol{t}$, and choice of $A(t)$. Plot in a single figure all components of $A(t)$. Plot the approximants, as well as f in a single figure. Construct a convergence plot of the approximations by representation of point data $\mathcal{E} = \{(\log n, \log \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|) | \, \boldsymbol{A} \in \mathbb{R}^{m \times n}, n \in \mathcal{N}\}$. For the largest value of n within $\mathcal{N}$, construct a figure superimposing increasing number of sampling points, $m \in \mathcal{M}$. Comment on what you observe in each individual exercise. Also compare results from the different exercises.

1. Start with the classical example due to Runge (1901)

$$
\begin{aligned}
&f\colon [-1,1] \to \mathbb{R},\, f(t) = \frac{1}{(1+25t^2)}, t_i = \frac{2(i-1)}{m-1} - 1,\\
&\mathcal{M} = \{16, 32, 64, 128, 256\}, \mathcal{N} = \{4, 8, 16, 32\},\\
&A(t) = \begin{bmatrix} 1 & t & t^2 & \ldots & t^{n-1} \end{bmatrix}.
\end{aligned}
$$

2. Instead of the equidistant point samples of the Runge example above use the Chebyshev nodes

$$
t_i = \cos\left(\frac{2i-1}{2m}\pi\right),
$$

keeping other parameters as in Problem 1.

3. Instead of the monomial family of the Runge example, use the Fourier basis

$$
A(t) = \begin{bmatrix} 1 & \cos \pi t & \sin \pi t & \ldots & \cos \pi n t & \sin \pi n t \end{bmatrix}
$$

keeping other parameters as in Problem 1. In this case $\boldsymbol{A} \in \mathbb{R}^{m \times (2n+1)}$.

4. Instead of the monomial family of the Runge example, use the piecewise linear B-spline basis

$$
A(t) = \begin{bmatrix} N_1(t) & N_2(t) & \ldots & N_n(t) \end{bmatrix},
$$

$$
N_i(t) = \begin{cases} 0, & t < t_{i-1} \\ \dfrac{t - t_{i-1}}{h} & t_{i-1} \leqslant t < t_i \\ \dfrac{t_{i+1} - t}{h} & t_i \leqslant t < t_{i+1} \\ 0 & t_{i+1} < t \end{cases}, h = \frac{2}{m-1},
$$

keeping other parameters as in Problem 1.

2 Track 2

1. If $\boldsymbol{Q} \in \mathbb{C}^{m \times n}$ has orthonormal columns, prove that $\boldsymbol{P_Q} = \boldsymbol{Q}\boldsymbol{Q}^*$ is an orthogonal projector onto $C(\boldsymbol{Q})$. Determine the expression of $\boldsymbol{P_A}$, the projector onto $C(\boldsymbol{A})$, with $\boldsymbol{A} \in \mathbb{C}^{m \times n}$. Compare the number of arithmetic operations required to compute $\boldsymbol{y} = \boldsymbol{P_A}\boldsymbol{x}$, by comparison to first determining the QR factorization, $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$, and then computing $\boldsymbol{y} = \boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{x}$.

   - Solution.

$\mathbf{P}$ is said to be an orthogonal projector if $\boldsymbol{P} = \boldsymbol{P}^*$ and $\boldsymbol{P}^2 = \boldsymbol{P}$ ([1]). If $\boldsymbol{Q} \in \mathbb{C}^{m\times n}$ has orthonormal columns, then it is an orthogonal matrix and therefore $\boldsymbol{Q}^* = \boldsymbol{Q}^{-1}$. Then $\boldsymbol{P_Q} = \boldsymbol{Q}\boldsymbol{Q}^* = (\boldsymbol{Q}\boldsymbol{Q}^*)^* = \boldsymbol{P_Q^*}$. If $\boldsymbol{P_Q}$ is an orthogonal projector onto the columns of $\boldsymbol{Q}$, then

$$\begin{aligned} \boldsymbol{P_Q Q} &= \boldsymbol{Q} \quad / \boldsymbol{Q}^* \cdot \\ \boldsymbol{Q}^* \boldsymbol{P_Q Q} &= \mathbb{I}, \quad \text{since } \boldsymbol{Q} \text{ is orthonormal} \\ \iff \boldsymbol{P_Q} &= \boldsymbol{Q}\boldsymbol{Q}^* \end{aligned}$$

Now if we want to determine the expression of $\boldsymbol{P_A}$, the projector onto $C(\boldsymbol{A})$, with $\boldsymbol{A} \in \mathbb{C}^{m\times n}$ a general matrix. We think about the orthogonal projection of a general vector $\boldsymbol{v}$ onto range$(\boldsymbol{A})$, and denote it by $\boldsymbol{y}$. Let a_j is the jth column of the matrix $\boldsymbol{A}$, then $a_j^*(y - v) = 0$ for each j, or equivalently $\boldsymbol{A^*}(\boldsymbol{y} - \boldsymbol{v}) = \boldsymbol{0}$. Now since $\boldsymbol{y} \in \text{range}(\boldsymbol{A})$, we can leverage that there exists $\boldsymbol{x}$ such that $\mathbf{A}\mathbf{x} = \boldsymbol{y}$. Then

$$\begin{aligned} \boldsymbol{A^*}(\boldsymbol{Ax} - \boldsymbol{v}) &= \boldsymbol{0} \\ \iff \boldsymbol{A^*Ax} &= \boldsymbol{A}^*\boldsymbol{v} \\ \iff \boldsymbol{x} &= (\boldsymbol{A^*A})^{-1}\boldsymbol{A}^*\boldsymbol{v} \end{aligned}$$

Now since $\boldsymbol{Ax} = \boldsymbol{y} \implies \boldsymbol{y} = \boldsymbol{A}(\boldsymbol{A^*A})^{-1}\boldsymbol{A}^*\boldsymbol{v} = \boldsymbol{P_A v}$. Whence

$$\boldsymbol{P_A} = \boldsymbol{A}(\boldsymbol{A^*A})^{-1}\boldsymbol{A}^*$$

The number of arithmetic operations required to calculate $\boldsymbol{y} = \boldsymbol{P_A x}$ is first $2mn^2$ to calculate $(\boldsymbol{A^*A})$, then for the inverse is $2n^3/3$ (with Gaussian elimination), then for the other two matrix multiplications we get $2mn^2 + 2mn^2$. Hence the total cost is $6mn^2 + \frac{2n^3}{3}$. The number of arithmetic operations for the QR factorization computed using the modified Gram-Schmidt algorithm is $2mn^2$ ([1]) and then a 2 matrix vector multiplication gives $4mn$, hence a total of $2mn^2 + 4mn$. Hence computing the QR factorization first and then projecting is more efficient.

2. Continuing Problem 1, determine $\|\boldsymbol{P_Q}\|_2$, and express $\|\boldsymbol{P_A}\|_2$ in terms of the singular value decomposition of $\boldsymbol{A}$. Comment the result, considering, say, length of shadows at various times of day.

   - Solution.

$$\begin{aligned} \|\boldsymbol{P_A}\|_2 &= \|\boldsymbol{A}(\boldsymbol{A^*A})^{-1}\boldsymbol{A}^*\|_2 \\ &= \|\boldsymbol{U\Sigma V^*}(\boldsymbol{V\Sigma^* U^* U\Sigma V^*})^{-1}\boldsymbol{V\Sigma^* U^*}\|_2 \\ &= \|\boldsymbol{U\Sigma V^*}(\boldsymbol{V\Sigma^*\ \Sigma V^*})^{-1}\boldsymbol{V\Sigma^* U^*}\|_2 \\ &= \|\boldsymbol{U\Sigma V^* V}(\boldsymbol{\Sigma\Sigma^*})^{-1}\boldsymbol{V^* V\Sigma^* U^*}\|_2 \\ &= \|\boldsymbol{U\Sigma}\,(\boldsymbol{\Sigma\Sigma^*})^{-1}\,\boldsymbol{\Sigma^* U^*}\|_2 \end{aligned}$$

Also we have that for an orthogonal matrix $\boldsymbol{V} \in \mathbb{C}^{m\times n}$ and a vector $\boldsymbol{x} \in \mathbb{C}^n$, $\|\boldsymbol{Vx}\| = \|\boldsymbol{Vx}\|$. with $\|\cdot\|$ any matrix norm. Hence $\|\boldsymbol{U\ U^*}\|_2 = \|\boldsymbol{U^*}\|_2$ and also

$$\begin{aligned} \|\boldsymbol{P_Q}\|_2 &= \|\boldsymbol{QQ^*}\|_2 \\ &= \|\boldsymbol{Q^*}\|_2 \end{aligned}$$

Both of the proyections are equivalent. We see that the norm of a projection depends on which space the vector $\boldsymbol{x}$ belongs. Remark that

$$\|\boldsymbol{x}\| = \|\boldsymbol{P_Q x}\| + \|(\mathbb{I} - \boldsymbol{P_Q})\boldsymbol{x}\| = \|\boldsymbol{P_Q x}\| + \|\boldsymbol{P_{Q^\perp} x}\|$$

For example we can take $\boldsymbol{P_Q} = \boldsymbol{Q}\boldsymbol{Q}^*$ as this is more insightful form of the proyector. Then the norm of the projection will depend mainly if $\boldsymbol{x} \in \text{span}(\boldsymbol{Q})$, in this case for example $\|\boldsymbol{P_Q x}\| = \|\boldsymbol{x}\|$. If $\boldsymbol{x} \in \text{span}(\boldsymbol{P_Q}\perp)$ then the norm $\|\boldsymbol{P_Q x}\| = 0$. So between this two cases there is a spectrum of intermediate cases.

Now for the general case of projections (not necesssarily orthogonal) we can also have the case that $\|\boldsymbol{P}\| \geq 1$. This is the case of the length of shadows at various times of day, for example at the sunset one could see a shadow bigger than ourselves because the projection is oblique. But when the sun is just above the length of the shadow it's 0 because we are not in the span of the projection.

3. A matrix $\boldsymbol{A} = [a_{ij}] \in \mathbb{C}^{m\times n}$ is said to be banded with bandwidth B if $a_{ij} = 0$ for $|i-j| > B$. Implement the modified Gram-Schmidt algorithm for $\boldsymbol{A} \in \mathbb{C}^{m\times n}$ a banded matrix with bandwidth B using as few arithmetic operations as possible.

   - Solution.

     First let's review the modified Gram-Schmidt algorithm

Algorithm (Gram-Schmidt)
Given n vectors $\boldsymbol{a}_1, ..., \boldsymbol{a}_n$
Initialize $\boldsymbol{q}_1 = \boldsymbol{a}_1, .., \boldsymbol{q}_n = \boldsymbol{a}_n$, $\boldsymbol{R} = \boldsymbol{I}_n$
for $i = 1$ to n
  $r_{ii} = (\boldsymbol{q}_i^T \boldsymbol{q}_i)^{1/2}$
  if $r_{ii} < \epsilon$ break;
  $\boldsymbol{q}_i = \boldsymbol{q}_i / r_{ii}$
  for $j = i+1$ to n
    $r_{ij} = \boldsymbol{q}_i^T \boldsymbol{a}_j$; $\boldsymbol{q}_j = \boldsymbol{q}_j - r_{ij}\boldsymbol{q}_i$
  end
end
return $\boldsymbol{Q}, \boldsymbol{R}$

```
∴ function mgs(A)
    m,n=size(A); Q=copy(A); R=zeros(n,n)
    for i=1:n
      R[i,i]=sqrt(Q[:,i]'*Q[:,i])
      if (R[i,i]<eps())
        break
      end
      Q[:,i]=Q[:,i]/R[i,i]
      for j=i+1:n
        R[i,j]=Q[:,i]'*A[:,j]
        Q[:,j]=Q[:,j]-R[i,j]*Q[:,i]
      end
    end
    return Q,R
  end;
```

```
∴
```

Remark that as $\boldsymbol{A} = [a_{ij}] \in \mathbb{C}^{m\times n}$ is banded with bandwidth B, then as $a_{ij} = 0$ for $|i-j| > B$ implies that some of the dot products and for loops could be simplified since the matrix is sparse. For example when $j > i+B$ (since j goes from $i+1$ to n in the inner loop), then a_{ij} is zero and this implies the inner for loop can be done more efficiently. Then the algorithm becomes

Algorithm (Gram-Schmidt banded)
Given n vectors $\boldsymbol{a}_1, ..., \boldsymbol{a}_n$
Initialize $\boldsymbol{q}_1 = \boldsymbol{a}_1, .., \boldsymbol{q}_n = \boldsymbol{a}_n$, $\boldsymbol{R} = \boldsymbol{I}_n$
for $i = 1$ to n
  $r_{ii} = (\boldsymbol{q}_i^T \boldsymbol{q}_i)^{1/2}$
  if $r_{ii} < \epsilon$ break;
  $\boldsymbol{q}_i = \boldsymbol{q}_i / r_{ii}$
  for $j = i+1$ to $i+B+1$
    $r_{ij} = \boldsymbol{q}_i^T \boldsymbol{a}_j$; $\boldsymbol{q}_j = \boldsymbol{q}_j - r_{ij}\boldsymbol{q}_i$
  end
end
return $\boldsymbol{Q}, \boldsymbol{R}$

```
∴ function mgsb(A,B)
    m,n=size(A); Q=copy(A); R=zeros(n,n)
    for i=1:n
      R[i,i]=sqrt(Q[:,i]'*Q[:,i])
      if (R[i,i]<eps())
        break
      end
      Q[:,i]=Q[:,i]/R[i,i]
      for j=i+1:min(i+B+1,n)
          R[i,j]=Q[:,i]'*A[:,j]
          Q[:,j]=Q[:,j]-R[i,j]*Q[:,i]
      end
    end
    return Q,R
  end;
```

```
∴
```

```
∴ function Bandmat(A,B)
  A=copy(A);
  m=size(A)[1];
  n=size(A)[2];
  for i=1:m
     for j=1:n
     if abs(i-j)>B
        A[i,j]=0
     end
     end
  end
  return A
  end
```

```
Bandmat
```

```
∴ B=1;A=rand(3,3)
```

$$\begin{bmatrix} 0.788201240252665 & 0.800702885511285 & 0.3620861532942121 \\ 0.03623830569412734 & 0.7908612002223709 & 0.9572458333691343 \\ 0.29918102203657027 & 0.7757522597991917 & 0.9005393912547199 \end{bmatrix} \tag{1}$$

```
∴ Ab=Bandmat(A,B)
```

$$\begin{bmatrix} 0.788201240252665 & 0.800702885511285 & 0.0 \\ 0.03623830569412734 & 0.7908612002223709 & 0.9572458333691343 \\ 0.0 & 0.7757522597991917 & 0.9005393912547199 \end{bmatrix} \tag{2}$$

```
∴ Q1,R1=mgs(Ab);
```

```
∴ Q2,R2=mgsb(Ab,B);
```

```
∴ print(Q1==Q2);print("\n"); print(norm(Q1-Q2,2))
```

```
true
0.0
```

```
∴ B=2;A=rand(4,4);
```

```
∴ Ab=Bandmat(A,B);
```

```
∴ Q1,R1=mgs(Ab);
```

```
∴ Q2,R2=mgsb(Ab,B);
```

```
∴ print(Q1==Q2);print("\n"); print(norm(Q1-Q2,2))
```

```
true
0.0
```

```
∴
```

Hence we can verify that the algorithm continues giving the QR decomposition but now it is much more efficient. It is worth mentioning that the product $(\boldsymbol{q}_i^T \boldsymbol{q}_i)$ could also be optimized but this is a sort of less important optimization.

4. Solve Problem 1, Track 1.

- Solution (Julia Code).

```
∴ m=256; n=32; h=2.0/(m-1); t=(0:m-1)*h .- 1; N=2 .^(2:5);
```

```
∴ function RungeB(m,n,t)
  A=ones(m,1);
  for j=1:n-1
    A = [A t .^ j]
  end
  return A
  end;
```

```
∴ clf();Basis=RungeB(m,maximum(N),t);cd("//Volumes//GoogleDrive//Mi␣unidad//Docs␣Drive//
```

```
∴ for nin in 1:8
      plot(t,Basis[:,nin]);
  end;
```

```
∴ grid("on");xlabel("t"),ylabel("A(t)");title("Monomials");
```

```
∴ savefig("H0301.eps");
```

```
∴ f(t)=1/(1+25*t^2);y=f.(t);error=zeros(length(N));
```

```
∴ close();figure(figsize=(8,6));s=2; i=1:s:m; ts=t[i];
  ys=f.(ts);plot(ts,ys,"ok");
```

```
∴ for (idx,nk) in enumerate(N)
      local x; local b; local bassis;
      basis=Basis[:,1:nk];
      x=basis\y;
      b=basis*x;
      error[idx]=norm((y.-b),2)/length(t)
      plot(t,b);
  end;
```

```
∴ labels=["f(t)␣sampled";"Approximant␣with␣n=4";"Approximant␣with␣n=8";"Approximant␣with
  legend(labels,loc="upper␣left"); xlabel("t");ylabel("A(t)");
  grid("on");title("Runge␣approximation␣of␣\$f(t)\$");
```

```
∴ savefig("H0302.eps");
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```

```
∴ clf();figure(figsize=(8,6));plot(log.(2,N),log.(2,error));
```

```
∴ xlabel("t");ylabel("\$\\mathcal{E}\$"); grid("on");title("Log-
  log␣plot␣of␣the␣error");
```

```
∴ savefig("H0303.eps");
```

```
∴ M=2 .^(4:8);n=2^5; close();figure(figsize=(8,6));s=2; i=1:s:m;
  ts=t[i]; ys=f.(ts);plot(ts,ys,"ok");
```

```
∴ for (idx,mk) in enumerate(M)
      local x; local b; local bassis; local h; local t; local y;
      h=2.0/(mk-1);
      t=(0:mk-1)*h .- 1;
      y=f.(t);
      basis=RungeB(mk,n,t);
      x=basis\y;
      b=basis*x;
      plot(t,b,linewidth=3);
  end;
```

```
∴ labels=["f(t) sampled";"Approximant with m=16";"Approximant with m=32";"Approximant wi
  legend(labels,loc="upper left"); xlabel("t");ylabel("A(t)");
  grid("on");title("Runge approximation of \$f(t)\$");
```

```
∴ savefig("H0304.eps");
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```

```
∴
```

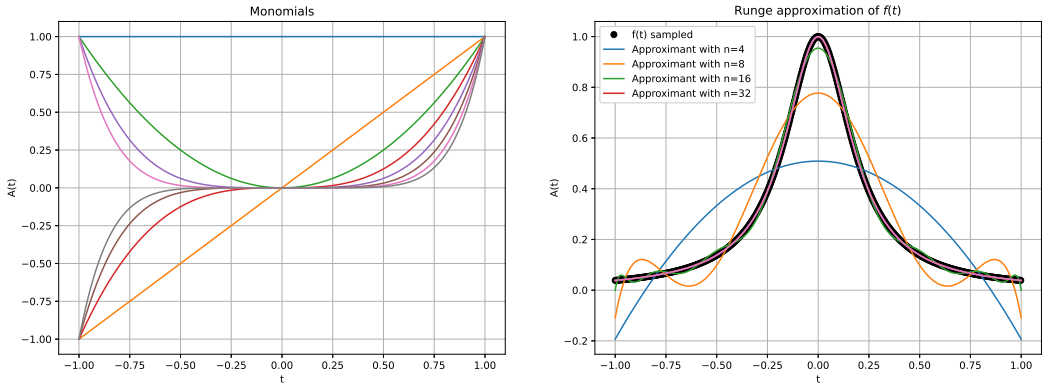

Figure 2. Components of $A(t)$, the Runge basis monomials (left). Approximants of $f(t)$ in conjunction with $f(t)$ sampled at the black dots (right).

Remark the basis of monomials have a similar shape all with the same root $x = 0$ (multiple for order large than $n = 1$). The basis only reaches a "good" approximation of $f(t)$ for $n = 32$. For $n = 4$ we have a sort of moving average of $f(t)$ and for $n = 8$ we get oscillations when the convexity of the function changes.

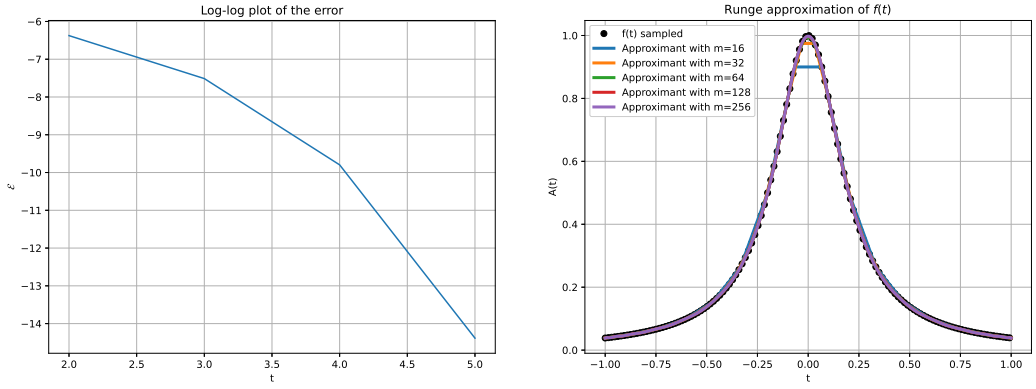

Figure 3. Convergence plot of the approximations by representation of point data $\mathcal{E} = \{(\log\, n,\ \log\, \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|) |\ \boldsymbol{A} \in \mathbb{R}^{m\times n},\ n \in \mathcal{N}\}$ (left). Superposition of increasing number of sampling points, $m \in \mathcal{M}$ with $n = 32$ (right).

We see that the error decreases about 2^8 times when N changes 2^3. For the increasing number of sample points we can note that the approximation of the peak of $f(t)$ has a considerable error when m is small.

5. Solve Problem 4, Track 1.

- Julia code.

```
∴ m=256; n=32; h=2.0/(m-1); t=(0:m-1)*h .- 1; N=2 .^(2:5);
```

```
∴ function BSpline(m,n,t);
  local A;
  A = zeros(m,n);
  h=2.0/(m-1);
  tm=t;
  hn = 2.0/(n-1);
  tn=(0:n-1)*hn .- 1
  h=2.0/(m-1);
  for j=1:n
     for i=1:m
       if tn[max(1,j-1)]<=tm[i]<tn[j]
          A[i,j]=(tm[i]-tn[max(1,j-1)])/h;
       elseif tn[j]<=tm[i]<tn[min(j+1,n)]
          A[i,j]=(tn[min(m,j+1)]-tm[i])/h;
       end
     end
  end
  return A
  end;
```

```
∴ Basis=BSpline(m,32,t);clf();cd("//Volumes//GoogleDrive//Mi␣unidad//Docs␣Drive//UNC//Se
```

```
∴ for nin in 1:n
      plot(t,Basis[:,nin]);
  end;
```

```
∴ grid("on");xlabel("t"),ylabel("A(t)");title("B-spline␣basis");
```

```
∴ savefig("H03P501.eps");
```

```
∴ f(t)=1/(1+25*t^2);y=f.(t);error=zeros(length(N));
```

```
∴ clf();figure(figsize=(8,6));s=2; i=1:s:m; ts=t[i];
  ys=f.(ts);plot(ts,ys,"ok");
```

```
∴ for (idx,nk) in enumerate(N)
      local x; local b; local bassis;
      basis=BSpline(m,nk,t);
      x=basis\y;
      b=basis*x;
      error[idx]=norm((y.-b),2)/length(t)
      plot(t,b);
  end;
```

```
∴ labels=["f(t)␣sampled";"Approximant␣with␣n=4";"Approximant␣with␣n=8";"Approximant␣with
  legend(labels,loc="upper␣left"); xlabel("t");ylabel("A(t)");
  grid("on");title("B-spline␣approximation␣of␣\$f(t)\$");
```

```
∴ savefig("H03P502.eps");
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```

```
∴ close();figure(figsize=(8,6));plot(log.(2,N),log.(2,error));
```

```
∴ xlabel("t");ylabel("\$\\mathcal{E}\$"); grid("on");title("Log-
  log␣plot␣of␣the␣error");
```

```
∴ savefig("H03P503.eps");
```

```
∴ M=2 .^(4:8);n=2^5; close();figure(figsize=(8,6));s=2; i=1:s:m;
  ts=t[i]; ys=f.(ts);plot(ts,ys,"ok");
```

```
∴ for (idx,mk) in enumerate(M)
      local x; local b; local bassis; local h; local t; local y;
      h=2.0/(mk-1);
      t=(0:mk-1)*h .- 1;
      y=f.(t);
      basis=BSpline(mk,n,t);
      x=basis\y;
      b=basis*x;
      plot(t,b,linewidth=3);
  end;
```

```
∴ labels=["f(t)␣sampled";"Approximant␣with␣m=16";"Approximant␣with␣m=32";"Approximant␣wi
  legend(labels,loc="upper␣left"); xlabel("t");ylabel("A(t)");
  grid("on");title("B-spline␣approximation␣of␣\$f(t)\$");
```

```
∴ savefig("H03P504.eps");
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```

```
∴
```

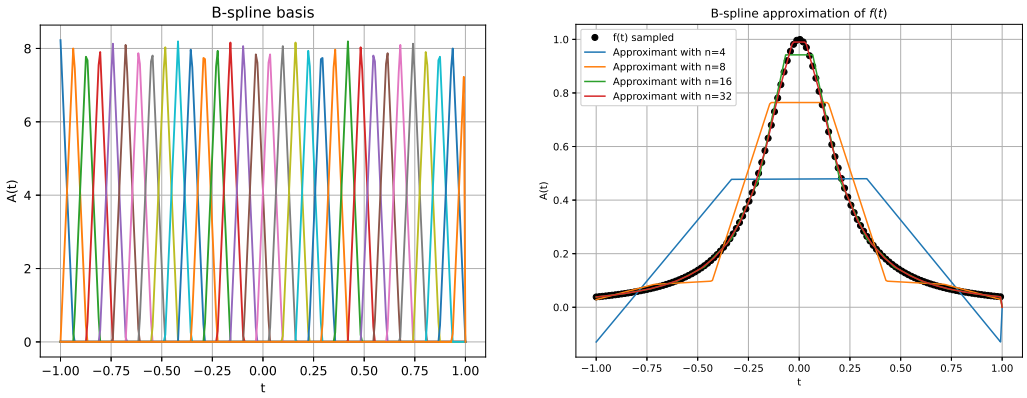

Figure 4. Components of $A(t)$, the B-spline basis (left). Approximants of $f(t)$ in conjunction with $f(t)$ sampled at the black dots (right).

Remark that the shape of this basis is much less smooth than any polynomial basis. Since the function to be approximated is a polynomial the approximation is less precise given it is non-smooth. However the approximation with a smaller bassis exhibits less oscillations

with respect to the monomial basis and the approximation at the beggining is better.

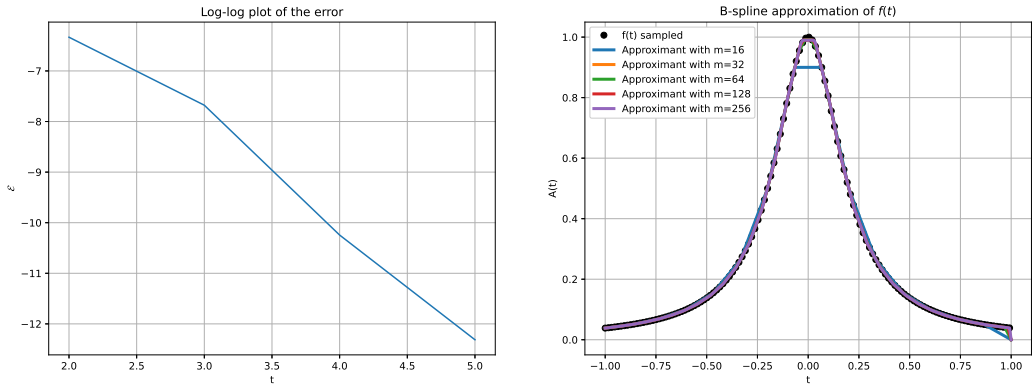

Figure 5. Convergence plot of the approximations by representation of point data $\mathcal{E} = \{(\log\, n,\ \log\, \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|) |\ \boldsymbol{A} \in \mathbb{R}^{m\times n},\ n \in \mathcal{N}\}$ (left). Superposition of increasing number of sampling points, $m \in \mathcal{M}$ with $n = 32$ (right).

We see that the error decreases about 2^6 times when N changes 2^3. Hence the approximation is not as good as the one with the monomial bassis for large n, but for small n we see that the error is less with the respect to the monomial basis. For the increasing number of sample points we can note that the main error is with $m = 16$ in the peak of $f(t)$ because the derivative is higher there.

6. In Problem 1, Track 1, replace the monomial basis with the Legendre polynomials, whose samples are determined by QR decomposition $\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{A}$. The resulting least squares problem is now

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \|\boldsymbol{y} - \boldsymbol{Q}\boldsymbol{x}\|_2 .$$

Solution.

- Julia code.

```
∴ m=256; n=32; h=2.0/(m-1); t=(0:m-1)*h .- 1; N=2 .^(2:5);
```

```
∴ function Legendre(m,n,t);
  local Bassis;
  M=RungeB(m,maximum(N),t);
  Q,R=qr(M);
  S=diagm(1.0 ./ Q[m,:]);
  Bassis=Q*S;
  return Bassis
  end
```

```
Legendre
```

```
∴ Bassis=Legendre(m,n,t);close();
```

```
∴ for nin in 1:8
      plot(t,Bassis[:,nin]);
  end;
```

```
∴ grid("on");xlabel("t"),ylabel("A(t)");title("Legendre␣polynomials");
```

```
∴ savefig("H03P601.eps");
```

```
∴ f(t)=1/(1+25*t^2);y=f.(t);error=zeros(length(N));
```

```
∴ clf();figure(figsize=(8,6));s=2; i=1:s:m; ts=t[i];
  ys=f.(ts);plot(ts,ys,"ok");
```

```
∴ for (idx,nk) in enumerate(N)
      local x; local b; local bassis;
      bassis=Bassis[:,1:nk];
      x=bassis\y;
      b=bassis*x;
      error[idx]=norm((y.-b),2)/length(t)
      plot(t,b);
  end;
```

```
∴ labels=["f(t)␣sampled";"Approximant␣with␣n=4";"Approximant␣with␣n=8";"Approximant␣with
  legend(labels,loc="upper␣left"); xlabel("t");ylabel("A(t)");
  grid("on");title("Legendre␣approximation␣of␣\$f(t)\$");
```

```
∴ savefig("H03P602.eps");
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```

```
∴ clf();figure(figsize=(8,6));plot(log.(2,N),log.(2,error));
```

```
∴ xlabel("t");ylabel("\$\\mathcal{E}\$"); grid("on");title("Log-
  log␣plot␣of␣the␣error");
```

```
∴ savefig("H03P603.eps");
```

```
∴ M=2 .^(4:8);n=2^5; close();figure(figsize=(8,6));s=2; i=1:s:m;
  ts=t[i]; ys=f.(ts);plot(ts,ys,"ok");
```

```
∴ for (idx,mk) in enumerate(M)
      local x; local b; local bassis; local h; local t; local y;
      h=2.0/(mk-1);
      t=(0:mk-1)*h .- 1;
      y=f.(t);
      bassis=Legendre(mk,n,t);
      x=bassis\y;
      b=bassis*x;
      plot(t,b,linewidth=3);
  end;
```

```
∴ labels=["f(t)␣sampled";"Approximant␣with␣m=16";"Approximant␣with␣m=32";"Approximant␣wi
  legend(labels,loc="upper␣left"); xlabel("t");ylabel("A(t)");
  grid("on");title("Legendre␣approximation␣of␣\$f(t)\$");
```

```
∴ savefig("H03P604.eps");
```

```
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
The PostScript backend does not support transparency; partially
transparent artists will be rendered opaque.
```

```
∴
```

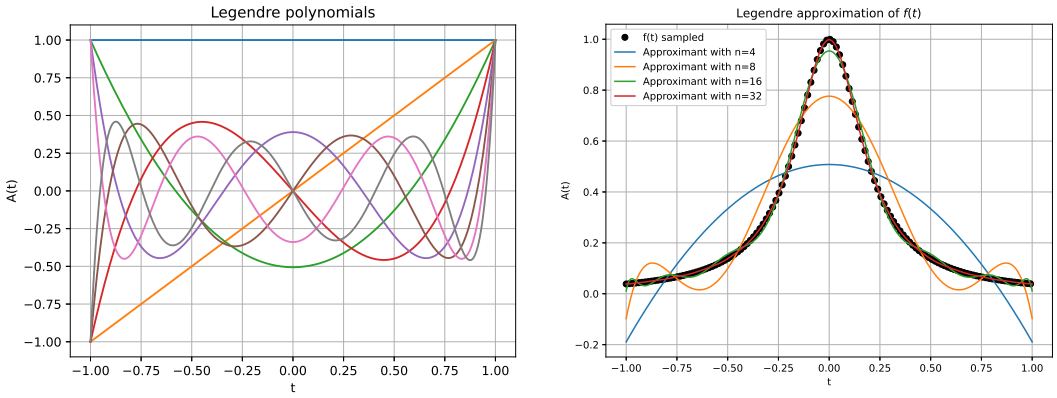

Figure 6. Components of $A(t)$, the Legendre polynomials bassis (left). Approximants of $f(t)$ in the Legendre bassis in conjunction with $f(t)$ sampled at the black dots (right).

We can easily see that the bassis has changed dramatically to the one of Runge monomials. However the approximants didn't change. This is due to the fact that taking the least squares still utilizes the same polynomial approximation in the end most probably.

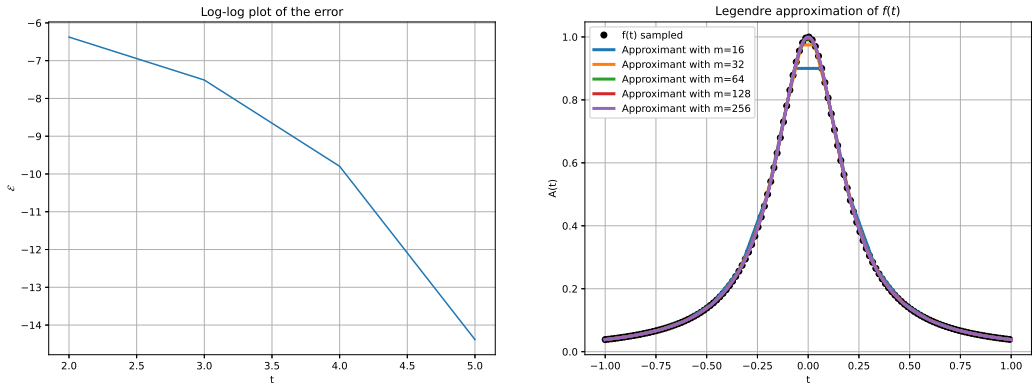

Figure 7. Convergence plot of the approximations by representation of point data $\mathcal{E} = \{(\log\, n,\ \log\, \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|)|\ \boldsymbol{A} \in \mathbb{R}^{m\times n},\ n \in \mathcal{N}\}$ (left). Superposition of increasing number of sampling points, $m \in \mathcal{M}$ with $n = 32$ (right).

We see that the error decreases about 2^8 times when N changes 2^3 also. For the increasing number of sample points we note that over $m = 64$ the approximation does not change.

Bibliography

[1] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.