MATH661 Homework 3 - Least squares problems

by Leyi Zhang

09/15/21

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

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

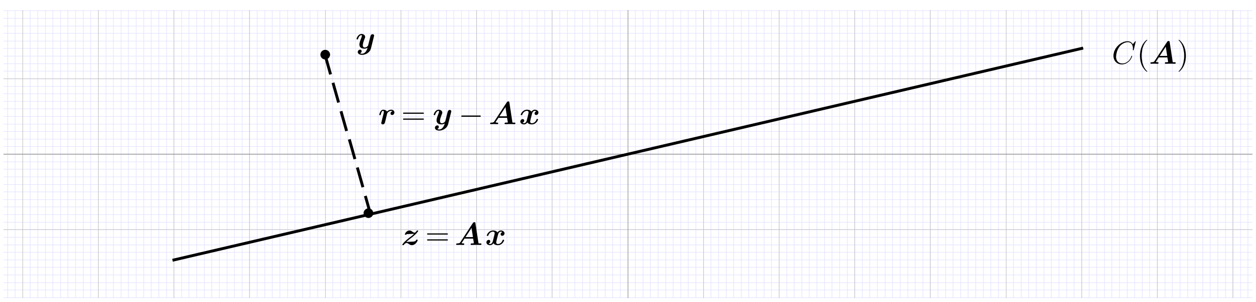

Figure 1. Least squares (2-norm error minimization) problem.

Track 1

Consider data $\mathcal{D} = \{(t_i, y_i) | i = 1, 2, \ldots, m\}$ 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)

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

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.

Problem 1.1

Start with the classical example due to Runge (1901)

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

Problem 1.2

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.

Problem 1.3

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

$$A(t) = [\ 1\ \ \cos \pi t\ \ \sin \pi t\ \ \ldots\ \ \cos \pi n t\ \ \sin \pi n t\ ]$$

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

Problem 1.4

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

$$A(t) = [\ N_1(t)\ \ N_2(t)\ \ \ldots\ \ N_n(t)\ ],$$

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

keeping other parameters as in Problem 1.

```
∴ using Plots; using LinearAlgebra
```

```
∴
```

Track 2

Problem 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:

Definition 1. *A orthogonal projector is a square matrix* $\boldsymbol{P}$ *that satisfies* $\boldsymbol{P}^2 = \boldsymbol{P}$ *and* $\boldsymbol{P}^* = \boldsymbol{P}$.

Let $\boldsymbol{Q} = [\boldsymbol{q}_1\, \boldsymbol{q}_2 \cdots \boldsymbol{q}_n] \in \mathbb{C}^{m \times n}$, where $\boldsymbol{q}_j = [q_{j,1}\, q_{j,2} \cdots q_{j,m}]^T$ is a column vector with $q_{j,i} \in \mathbb{C}$ for every $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$. Then $\boldsymbol{Q}^* = [\boldsymbol{q}_1\, \boldsymbol{q}_2 \cdots \boldsymbol{q}_n]^* \in \mathbb{C}^{n \times m}$. If we write them out, we have

$$\boldsymbol{Q} = [\boldsymbol{q}_1\, \boldsymbol{q}_2 \cdots \boldsymbol{q}_n] = \begin{bmatrix} q_{1,1} & q_{2,1} & \cdots & q_{n,1} \\ q_{1,2} & q_{2,2} & & \vdots \\ \vdots & & \ddots & \vdots \\ q_{1,m} & \cdots & \cdots & q_{n,m} \end{bmatrix},$$

$$\boldsymbol{Q}^* = \begin{bmatrix} \boldsymbol{q}_1^* \\ \boldsymbol{q}_2^* \\ \vdots \\ \boldsymbol{q}_n^* \end{bmatrix} = \begin{bmatrix} \overline{q_{1,1}} & \overline{q_{1,2}} & \cdots & \overline{q_{1,m}} \\ \overline{q_{2,1}} & \overline{q_{2,2}} & & \vdots \\ \vdots & & \ddots & \vdots \\ \overline{q_{n,1}} & \cdots & \cdots & \overline{q_{n,m}} \end{bmatrix}.$$

First, we want to show that $\boldsymbol{P}_{\boldsymbol{Q}}^* = \boldsymbol{P}_{\boldsymbol{Q}}$. We know that for any matrices $\boldsymbol{A}$, $\boldsymbol{B}$, we have $(\boldsymbol{A}\boldsymbol{B})^* = \boldsymbol{B}^*\boldsymbol{A}^*$. Thus

$$\boldsymbol{P}_{\boldsymbol{Q}}^* = (\boldsymbol{Q}\,\boldsymbol{Q}^*)^* = (\boldsymbol{Q}^*)^*\boldsymbol{Q}^* = \boldsymbol{Q}\,\boldsymbol{Q}^* = \boldsymbol{P}_{\boldsymbol{Q}}.$$

Then we want to show that $\boldsymbol{P}_{\boldsymbol{Q}}^2 = \boldsymbol{P}_{\boldsymbol{Q}}$. We are given that $\boldsymbol{P}_{\boldsymbol{Q}} = \boldsymbol{Q}\,\boldsymbol{Q}^*$, so we have

$$\boldsymbol{P}_{\boldsymbol{Q}}^2 = \boldsymbol{Q}\,\boldsymbol{Q}^*\boldsymbol{Q}\,\boldsymbol{Q}^*.$$

Consider

$$\boldsymbol{Q}^*\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{q}_1^* \\ \boldsymbol{q}_2^* \\ \vdots \\ \boldsymbol{q}_n^* \end{bmatrix} [\boldsymbol{q}_1\,\boldsymbol{q}_2 \cdots \boldsymbol{q}_n] = \begin{bmatrix} q_1^*q_1 & q_1^*q_2 & \cdots & q_1^*q_n \\ q_2^*q_1 & q_2^*q_2 & & \vdots \\ \vdots & & \ddots & \vdots \\ q_n^*q_1 & \cdots & \cdots & q_n^*q_n \end{bmatrix} \in \mathbb{C}^{n\times n}.$$

Since $\boldsymbol{Q}$ has orthonormal columns, we know that $\boldsymbol{q}_i^*\boldsymbol{q}_j = \delta_{ij}$ for every $i, j \in \{1, ..., n\}$. That gives us

$$\boldsymbol{Q}^*\boldsymbol{Q} = \boldsymbol{I}_n,$$

and we have

$$\boldsymbol{P}_{\boldsymbol{Q}}^2 = \boldsymbol{Q}\,\boldsymbol{Q}^*\boldsymbol{Q}\,\boldsymbol{Q}^* = \boldsymbol{Q}\,(\boldsymbol{Q}^*\boldsymbol{Q})\,\boldsymbol{Q}^* = \boldsymbol{Q}\,\boldsymbol{I}_n\boldsymbol{Q}^* = \boldsymbol{Q}\,\boldsymbol{Q}^* = \boldsymbol{P}.$$

Therefore, we conclude taht $\boldsymbol{P}_{\boldsymbol{Q}}$ is indeed an orthogonal projector.

Suppose we have a matrix $\boldsymbol{A} = [\boldsymbol{a}_1\,\boldsymbol{a}_2 \cdots \boldsymbol{a}_n] \in \mathbb{C}^{m\times n}$, where $\boldsymbol{a}_j = [a_{j,1}\,a_{j,2} \cdots a_{j,m}]^T$ with $a_{j,i} \in \mathbb{C}$ for every $i \in \{1, ..., m\}$ and $j \in \{1, ..., n\}$. Then $\boldsymbol{A}^* = [\boldsymbol{a}_1\ \boldsymbol{a}_2 \cdots \boldsymbol{a}_n]^* \in \mathbb{C}^{\boldsymbol{n}\times\boldsymbol{m}}$. That means $\boldsymbol{P}_{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{A}^* \in \mathbb{C}^{m\times m}$ such that

$$\boldsymbol{P}_{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{A}^* = \begin{bmatrix} \sum_{j=1}^n \overline{a_{j,1}} \cdot a_{j,1} & \sum_{j=1}^n \overline{a_{j,2}} \cdot a_{j,1} & \cdots & \sum_{j=1}^n \overline{a_{j,m}} \cdot a_{j,1} \\ \sum_{j=1}^n \overline{a_{j,1}} \cdot a_{j,2} & \sum_{j=1}^n \overline{a_{j,2}} \cdot a_{j,2} & & \vdots \\ \vdots & & \ddots & \vdots \\ \sum_{j=1}^n \overline{a_{j,1}} \cdot a_{j,m} & \cdots & \cdots & \sum_{j=1}^n \overline{a_{j,m}} \cdot a_{j,m} \end{bmatrix}.$$

$$\begin{aligned} \boldsymbol{P}_{\boldsymbol{A}} = \boldsymbol{A}\boldsymbol{A}^* &= \begin{bmatrix} \sum_{j=1}^n \overline{a_{j,1}} \cdot a_{j,1} & \sum_{j=1}^n \overline{a_{j,2}} \cdot a_{j,1} & \cdots & \sum_{j=1}^n \overline{a_{j,m}} \cdot a_{j,1} \\ \sum_{j=1}^n \overline{a_{j,1}} \cdot a_{j,2} & \sum_{j=1}^n \overline{a_{j,2}} \cdot a_{j,2} & & \vdots \\ \vdots & & \ddots & \vdots \\ \sum_{j=1}^n \overline{a_{j,1}} \cdot a_{j,m} & \cdots & \cdots & \sum_{j=1}^n \overline{a_{j,m}} \cdot a_{j,m} \end{bmatrix} \\ &= \left[\sum_{j=1}^n \overline{a_{j,1}}\boldsymbol{a}_j \quad \sum_{j=1}^n \overline{a_{j,2}}\boldsymbol{a}_j \quad \cdots \quad \sum_{j=1}^n \overline{a_{j,m}}\boldsymbol{a}_j \right]. \end{aligned}$$

For every entry of $\boldsymbol{P_A}$, we will need to carry our n multiplications and $n-1$ additions, i.e. $2n-1$ operations in total. Since $\boldsymbol{P_A}$ is a $m\times m$ matrix, it has m^2 entries. Thus we will need to carry out $m^2(2n-1)$ arithmetic operations to compute $\boldsymbol{P_A}$. Now suppose $\boldsymbol{x}\in\mathbb{C}^m$, then we need another m multiplications and $m-1$ additions to compute each of the m entries of $\boldsymbol{P_A}\boldsymbol{x}$. Therefore, the total number of operations to compute $\boldsymbol{P_A}\boldsymbol{x}$ will be

$$m^2(2n-1)+m(2m-1)=m^2(2n+1)-m.$$

Assume we are using the modified Gram-Schmidt process to carry out the QR decomposition of A. Consider $\boldsymbol{q}_i=\boldsymbol{a}_i$ with length m, one of the n columns of $\boldsymbol{A}$. We first need to compute its norm $\boldsymbol{q}_i^*\boldsymbol{q}_i$, which will take m multiplications, $m-1$ additions, and m square roots, so $3m-1$ operations. Then we normalize $\boldsymbol{q}_i$ by dividing each entry by its norm, and that takes m operations. Then for each $j\in\{i+1,\ldots,n\}$, we will compute $\boldsymbol{R}_{\mathrm{ij}}$ with a dot product $\boldsymbol{q}_i^*\boldsymbol{a}_j$ with m nultiplications and $m-1$ additions, and then compute $\boldsymbol{q}_j$ with a dot product $\boldsymbol{r}_i^*\boldsymbol{q}_j$ with m multiplications and $m-1$ additions, and then subtract the dot product from each entry of $\boldsymbol{q}_j$ with m operations. That is, for each j, we need $(2m-1)+(3m-1)=5m-2$ operations, and there are $n-i-1$ of j's for each i from 1 to $n-1$. Thus for each i from 1 to $n-1$, we need $(4m-1)+(5m-2)(n-i-1)$ operations, and for $i=n$, only $4m-1$ operations are needed.

Therefore, the total number of operations we will need for the modified Gram-Schmidt process is

$$(4m-1)n+\sum_{i=1}^{n-1}(5m-2)(n-i-1)=(4m-1)n+(5m-2)(n-2)(n-1).$$

Now we know that $\boldsymbol{Q}^*\in\mathbb{C}^{n\times m}$, $\boldsymbol{Q}\in\mathbb{C}^{m\times n}$, and $\boldsymbol{x}\in\mathbb{C}^m$. So each of the n entries of $\boldsymbol{Q}^*\boldsymbol{x}$ will take m multiplications and $m-1$ additions to compute, so $n(2m-1)$ in total for $\boldsymbol{Q}^*\boldsymbol{x}$. Then for each of the m entries of $\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{x}$, we will need n multiplications and $n-1$ additions, so $m(2n-1)$ in total for $\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{x}$. Thus we need $n(2m-1)+m(2n-1)=4mn-m-n$ operations to compute $\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{x}$.

In the end, the number of operations we need to first determine the QR decomposition of A and then compute $\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{x}$ would be

$$(5m-2)n^2-14mn+9m+4n-4.$$

Thus the cost of computing $\boldsymbol{P_A}\boldsymbol{x}$ directly would be of order m^2n, while the cost of computing $\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{x}$ after QR decomposition would be of order mn^2. The QR decomposition method will be much more efficient in the situations where we have $m\gg n$, which is often the case.

Problem 2.2

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:

Definition 2. *The 2-norm of a matrix* $\boldsymbol{A}\in\mathbb{C}^{m\times n}$ *is defined to be*

$$\|\boldsymbol{A}\|_2=\sup_{\boldsymbol{x}\in\mathbb{C}^n,\|\boldsymbol{x}\|_2=1}\|\boldsymbol{A}\boldsymbol{x}\|_2.$$

Since $\boldsymbol{P_Q}=\boldsymbol{P_Q^*}$ and $\boldsymbol{P_Q}=\boldsymbol{P_Q^2}$, for any $\boldsymbol{x}\in\mathbb{C}^m$, we have

$$\|\boldsymbol{P_Q}\boldsymbol{x}\|_2^2=(\boldsymbol{P_Q}\boldsymbol{x})^*\boldsymbol{P_Q}\boldsymbol{x}=\boldsymbol{x}^*\boldsymbol{P_Q^*}\boldsymbol{P_Q}\boldsymbol{x}=\boldsymbol{x}^*\boldsymbol{P_Q^2}\boldsymbol{x}=\boldsymbol{x}^*\boldsymbol{P_Q}\boldsymbol{x}.$$

Consider $\boldsymbol{Q}\colon\mathbb{C}^n\to\mathbb{C}^m$. We know that the codomain $\mathbb{C}^m=C(\boldsymbol{Q})\oplus N(\boldsymbol{Q}^*)$. That means any vector $\boldsymbol{x}\in\mathbb{C}^m$, can be written as $\boldsymbol{x}=\boldsymbol{v}+\boldsymbol{w}$ with unique vectors $\boldsymbol{v}\in C(\boldsymbol{Q})$ and $\boldsymbol{w}\in N(\boldsymbol{Q}^*)$.

Since $\boldsymbol{v}$ is an element in $C(\boldsymbol{Q})$, there exists some $\boldsymbol{y}\in\mathbb{C}^n$, such that $\boldsymbol{v}=\boldsymbol{Q}\boldsymbol{y}$. Since $\boldsymbol{P_Q}=\boldsymbol{Q}\boldsymbol{Q}^*$, we get $\boldsymbol{P_Q}\boldsymbol{v}=\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{Q}\boldsymbol{y}$. We have shown earlier that $\boldsymbol{Q}^*\boldsymbol{Q}=\boldsymbol{I}_n$, so $\boldsymbol{P_Q}\boldsymbol{v}=\boldsymbol{Q}\,\boldsymbol{y}=\boldsymbol{v}$.

Since $\boldsymbol{w}$ is an element in $N(\boldsymbol{Q}^*)$, we have $\boldsymbol{Q}^*\boldsymbol{w}=0$ by definition. Thus we get $\boldsymbol{P_Q}\boldsymbol{w}=\boldsymbol{Q}\boldsymbol{Q}^*\boldsymbol{w}=0$.

Therefore, we have

$$\begin{aligned}\|\boldsymbol{P_Q}\boldsymbol{x}\|_2^2=\boldsymbol{x}^*\boldsymbol{P_Q}\boldsymbol{x} &= (\boldsymbol{v}+\boldsymbol{w})^*\boldsymbol{P_Q}(\boldsymbol{v}+\boldsymbol{w})\\ &= (\boldsymbol{v}^*+\boldsymbol{w}^*)(\boldsymbol{P_Q}\boldsymbol{v}+\boldsymbol{P_Q}\boldsymbol{w})\\ &= (\boldsymbol{v}^*+\boldsymbol{w}^*)\boldsymbol{v}\\ &= \boldsymbol{v}^*\boldsymbol{v}+\boldsymbol{w}^*\boldsymbol{v}\end{aligned}$$

We know that $C(\boldsymbol{Q})\perp N(\boldsymbol{Q}^*)$, so we have $\boldsymbol{w}^*\boldsymbol{v}=0$, and

$$\|\boldsymbol{P_Q}\boldsymbol{x}\|_2^2=\boldsymbol{v}^*\boldsymbol{v}=\|\boldsymbol{v}\|_2^2.$$

Since $\boldsymbol{x}=\boldsymbol{v}+\boldsymbol{w}$ and $\|\boldsymbol{x}\|_2=1$,

$$\begin{aligned}1=\|\boldsymbol{x}\|_2^2 &= (\boldsymbol{v}+\boldsymbol{w})^*(\boldsymbol{v}+\boldsymbol{w})\\ &= (\boldsymbol{v}^*+\boldsymbol{w}^*)(\boldsymbol{v}+\boldsymbol{w})\\ &= \boldsymbol{v}^*\boldsymbol{v}+\boldsymbol{v}^*\boldsymbol{w}+\boldsymbol{w}^*\boldsymbol{v}+\boldsymbol{w}^*\boldsymbol{w}\\ &= \|\boldsymbol{v}\|_2^2+\|\boldsymbol{w}\|_2^2.\end{aligned}$$

That means we have $\|\boldsymbol{v}\|_2^2\leq 1$ with $\|\boldsymbol{v}\|_2^2=1$ when $\|\boldsymbol{w}\|_2^2=0$. Thus we conclude that

$$\sup_{\boldsymbol{x}\in\mathbb{C}^n,\|\boldsymbol{x}\|_2=1}\|\boldsymbol{P_Q}\boldsymbol{x}\|_2^2=\sup_{\boldsymbol{x}\in\mathbb{C}^n,\|\boldsymbol{x}\|_2=1}\|\boldsymbol{v}\|_2^2=1,$$

and

$$\|\boldsymbol{P_Q}\|_2=\sup_{\boldsymbol{x}\in\mathbb{C}^n,\|\boldsymbol{x}\|_2=1}\|\boldsymbol{P_Q}\boldsymbol{x}\|_2=\sqrt{\sup_{\boldsymbol{x}\in\mathbb{C}^n,\|\boldsymbol{x}\|_2=1}\|\boldsymbol{P_Q}\boldsymbol{x}\|_2^2}=1.$$

Suppose there exist matrices $\boldsymbol{U} \in \mathbb{C}^{m\times m}$, $\boldsymbol{V} \in \mathbb{C}^{n\times n}$ and $\boldsymbol{\Sigma} \in \mathbb{C}^{m\times n}$ with $\boldsymbol{U}^*\boldsymbol{U} = \boldsymbol{I}_m$ and $\boldsymbol{V}^*\boldsymbol{V} = \boldsymbol{I}_n$, such that

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*.$$

Then

$$\begin{aligned}\boldsymbol{A}\boldsymbol{A}^* &= (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*)(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*)^* \\ &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^*\boldsymbol{V}\boldsymbol{\Sigma}^*\boldsymbol{U}^* \\ &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^*\boldsymbol{U}^*,\end{aligned}$$

and we have

$$\begin{aligned}\|\boldsymbol{P}_{\boldsymbol{A}}\boldsymbol{x}\|_2^2 &= (\boldsymbol{P}_{\boldsymbol{A}}\boldsymbol{x})^*\boldsymbol{P}_{\boldsymbol{A}}\boldsymbol{x} = \boldsymbol{x}^*\boldsymbol{P}_{\boldsymbol{A}}^*\boldsymbol{P}_{\boldsymbol{A}}\boldsymbol{x} = \boldsymbol{x}^*\boldsymbol{P}_{\boldsymbol{A}}^2\boldsymbol{x} \\ &= \boldsymbol{x}^*\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^*\boldsymbol{U}^*\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^*\boldsymbol{U}^*\boldsymbol{x} \\ &= \boldsymbol{x}^*\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^*\boldsymbol{\Sigma}\boldsymbol{\Sigma}^*\boldsymbol{U}^*\boldsymbol{x} \\ &= \boldsymbol{x}^*\boldsymbol{U}(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^*)^2\boldsymbol{U}^*\boldsymbol{x}.\end{aligned}$$

In this problem, we found that the projection, or the "shadow", of a vector on the the comlumn space of a matrix cannot be longer than itself, but in real life the length of the shadow can have arbitrary length depending on the angle of the incoming light. I guess in this case, the projection is more similar to the shadow created by a light source that is directly above the object, like the shadow of an object on the equator at noon.

Problem 2.3

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.

(I don't have a working solution for this problem yet. I will read more and work on it over the next week.)

If $B < 0$, then $\boldsymbol{A}$ would be a matrix of zeros, so we only consider the case where $B \geq 0$. Let $b \in \mathbb{N}$ be the largest natural number such that $b \leq B$, i.e. we have $b \leq B < b + 1$. If $|i - j| > B$ for any i, j, then $|i - j| \geq b + 1$.

Suppose $\boldsymbol{A} = [\boldsymbol{a}_1\, \boldsymbol{a}_2 \cdots \boldsymbol{a}_n]$ where $\boldsymbol{a}_j = [a_{j,1}\, a_{j,2} \cdots a_{j,m}]$ with $a_{j,k} \in \mathbb{C}$ for all $j \in \{1, ..., n\}$ and $k \in \{1, ..., m\}$. Then for each j, we have $a_{j,1=\cdots=}\, a_{j,j-b-1} = a_{j,j+b+1} = \cdots = a_{j,m}{=}0$, i.e. there are at most $2b + 1$ nonzero entries in each $\boldsymbol{a}_j$.

For any $\boldsymbol{a}_j$ with $j < b + 1$, we only need to consider $j + b$ entries. For any $\boldsymbol{a}_j$ with $j > m - b$, we need to consider $m + b - j$ entries. For $\boldsymbol{a}_j$ with $b + 1 \leq j \leq m - b$, we need to consider $2b + 1$ entries.

Problem 2.4

(Solve Problem 1, Track 1.) Start with the classical example due to Runge (1901)

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

Julia (1.6.2) session in GNU TeXmacs

```
∴ function runge_f(t)
      f_t =  1 ./ (25*(t .^2) .+1)
      return f_t
  end
```

runge_f

```
∴ function monomial(n,m)
      h = 2.0/(m-1)
      t = [(i*h-1) for i=0:(m-1)]
      A = ones(m,1)
      for i=1:n-1
           A = [A (t .^ i)]
      end
      return A
  end
```

monomial

```
∴ function monomial_coef(n,m)
      h = 2.0/(m-1)
      t = [(i*h-1) for i=0:(m-1)]
      A = monomial(n,m)
      y = runge_f.(t)
      x = A\y
      return x
  end
```

monomial_coef

```
∴ M = 512; H = 2.0/(M-1); T = [(i*H-1) for i=0:(M-1)]; Y = runge_f(T);
```

```
∴ m_values = 2 .^ collect(4:8); m_size = length(m_values); err =
  Array{Float64}(undef, m_size);
```

```
∴
```

```
∴ for i=1:4
      n = 2^(i+1)
      m=96
      h = 2.0/(m-1)
      t = [(i*h-1) for i=0:(m-1)]
      A = ones(m,1)
      for i=1:n-1
           A = [A (t .^ i)]
      end
      display(Plots.plot(A, legend=false))
  end
```

```
┌ Warning: Assignment to `n` in soft scope is ambiguous because a
global variable by the same name exists: `n` will be treated as a new
local. Disambiguate by using `local n` to suppress this warning or
`global n` to assign to the existing global variable.
└ @ none:2
```

```
Base.Meta.ParseError("extra␣token␣after␣end␣of␣expression")
```

```
∴
```

```
∴ n=4; B=monomial(n,M); Approx=zeros(M,m_size);
```

```
∴ for i=1:m_size
      m=m_values[i];
      x=monomial_coef(n,m);
      Approx[:,i]=B*x;
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"));
```

```
∴
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.2464832341748203
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
1.1570229053972054
```

```
∴
```

```
∴ n=8; B=monomial(n,M); Approx=zeros(M,m_size);
```

```
∴ for i=1:m_size
      m=m_values[i];
      x=monomial_coef(n,m);
      Approx[:,i]=B*x;
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"));
```

```
∴
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.40277320620291507
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
1.5031719274633308
```

```
∴
```

```
∴ n=16; B=monomial(n,M); Approx=zeros(M,m_size);
```

```
∴ for i=1:m_size
      m=m_values[i];
      x=monomial_coef(n,m);
      Approx[:,i]=B*x;
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"));
```

```
∴
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.22324262284808913
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
0.41982876451361223
```

```
∴
```

```
∴ n=32; B=monomial(n,M); Approx=zeros(M,m_size);
```

```
∴ for i=1:m_size
      m=m_values[i];
      x=monomial_coef(n,m);
      Approx[:,i]=B*x;
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"));
```

```
∴
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
-1.2579998394391712
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
5.536127644117644
```

```
∴
```

Problem 2.5

(Solve Problem 4, Track 1.) Instead of the monomial family of the Runge example, use the piecewise linear B-spline basis

$$A(t) = [\ N_1(t) \quad N_2(t) \quad \dots \quad N_n(t)\ ],$$

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

keeping other parameters as in Problem 1.

(The plots I got seem quite strange but I'm not sure why that would be the case.)

```
∴ function bspline(n,m,M)
        h_m = 2.0/(m-1)
        h_M = 2.0/(M-1)
        t = [(i*h_m-1) for i=0:(m+1)]
        B = zeros(M,n)
        for i=1:n
            j_min = Int(max(ceil( (i-1)*h_m/h_M ),1))
            j_med = Int(ceil( i*h_m/h_M )-1)
            j_max = Int(min(ceil( (i+1)*h_m/h_M )-1,M))
            for j=j_min:Int(min(j_med,M))
                B[j,i] = (j*h_M-1-t[i])/h_m
            end
            if j_med+1 < M
                for j=(j_med+1):j_max
                    B[j,i] = (t[i+2]-j*h_M+1)/h_m
                end
            end
        end
        return B
    end
```

```
bspline
```

Julia (1.6.2) session in GNU TeXmacs

```
∴ # an alternative way to define the bspline basis
  function bspline_alt(n,m,M)
      p = max(n,m)
      h_m = 2.0/(m-1)
      h_M = 2.0/(M-1)
      t = [(i*h_m-1) for i=0:(p+1)]
      T = [(i*h_M-1) for i=0:(M+1)]
      B = zeros(M,n)
      for k=1:n
          for i=1:M
              for j=1:m
                  if T[i+1] >= t[k] && T[i+1] < t[k+1]
                      B[i,k]=(T[i+1]-t[k])/h_m
                  elseif T[i+1] >= t[k+1] && T[i+1] < t[k+2]
                      B[i,k]=(t[k+2]-T[i+1])/h_m
                  end
              end
          end
      end
      return B
  end
```

bspline_alt

```
∴
```

```
∴ function bspline_coef(n,m)
      h = 2.0/(m-1)
      t = [(i*h-1) for i=0:(m-1)]
      A = bspline_alt(n,m,m)
      y = runge_f.(t)
      x = A\y
      return x
  end
```

bspline_coef

```
∴
```

```
∴ for i=1:4
      local n = 2^(i+1)
      m=96
      h = 2.0/(m-1)
      t = [(i*h-1) for i=0:(m-1)]
      A = bspline_alt(n,m,m)
      display(Plots.plot(A, legend=false))
  end
```

UndefVarError(:Plots)

```
∴ B = bspline(16,16,512); Plots.plot(B)
```

```
Plot{Plots.GRBackend() n=16}
```

```
∴
```

```
∴ M = 512; H = 2.0/(M-1); T = [(i*H-1) for i=0:(M-1)]; Y = runge_f(T);
```

```
∴ m_values = 2 .^ collect(4:8); m_size = length(m_values); err =
  Array{Float64}(undef, m_size);
```

```
∴
```

```
∴ n = 4; Approx = zeros(M,m_size);
```

```
∴ for i=1:m_size
      m = m_values[i]
      x = bspline_coef(n,m)
      local B = bspline(n,m,M)
      Approx[:,i] = B*x
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"))
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The order of convergence is estimated to be","\n",q)
```

```
The order of convergence is estimated to be
0.2464832341748203
```

```
∴ print("The rate of convergence is","\n",s)
```

```
The rate of convergence is
1.1570229053972054
```

```
∴
```

```
∴ n = 8; Approx = zeros(M,m_size);
```

```
∴ for i=1:m_size
      m = m_values[i]
      x = bspline_coef(n,m)
      local B = bspline(n,m,M)
      Approx[:,i] = B*x
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"))
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.2464832341748203
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
1.1570229053972054
```

```
∴
```

```
∴ n = 16; Approx = zeros(M,m_size);
```

```
∴ for i=1:m_size
      m = m_values[i]
      x = bspline_coef(n,m)
      local B = bspline(n,m,M)
      Approx[:,i] = B*x
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"))
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.2464832341748203
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
1.1570229053972054
```

```
∴
```

```
∴ n = 32; Approx = zeros(M,m_size);
```

```
∴ for i=1:m_size
      m = m_values[i]
      x = bspline_coef(n,m)
      local B = bspline(n,m,M)
      Approx[:,i] = B*x
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"))
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.2464832341748203
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
1.1570229053972054
```

```
∴
```

Problem 2.6

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

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

```
∴ function legendre(n,m)
      h = 2.0/(m-1)
      t = [(i*h-1) for i=0:m-1]
      A = monomial(n,m)
      Q,R = qr(A)
      S = diagm(1.0 ./ Q[m,:])
      P = Q*S
      return P[:,1:n]
  end
```

```
legendre
```

```
∴ function legendre_coef(n,m)
      h = 2.0/(m-1)
      t = [(i*h-1) for i=0:(m-1)]
      A = legendre(n,m)
      y = runge_f.(t)
      x = A\y
      return x
  end
```

```
legendre_coef
```

```
∴
```

```
∴ M = 512; H = 2.0/(M-1); T = [(i*H-1) for i=0:(M-1)]; Y = runge_f(T);
```

```
∴ m_values = 2 .^ collect(4:8); m_size = length(m_values); err =
  Array{Float64}(undef, m_size);
```

```
∴ for i=1:3
      local n = 2^(i+1)
      m=96
      h = 2.0/(m-1)
      t = [(i*h-1) for i=0:(m-1)]
      P = legendre(n,m)
      display(Plots.plot(P, legend=false))
  end
```

```
∴
```

```
∴ n = 4; B = legendre(n,M); Approx = zeros(M,m_size);
```

```
∴ for i=1:m_size
      m = m_values[i]
      x = legendre_coef(n,m)
      Approx[:,i] = B*x
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"))
```

```
∴
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.2408324715806308
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
3.058988236991122
```

```
∴
```

```
∴ n = 8; B = legendre(n,M); Approx = zeros(M,m_size);
```

```
∴ for i=1:m_size
      m = m_values[i]
      x = legendre_coef(n,m)
      Approx[:,i] = B*x
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"))
```

```
∴
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.38479479155555435
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
1.521563387971189
```

```
∴
```

```
∴ n = 16; B = legendre(n,M); Approx = zeros(M,m_size);
```

```
∴ for i=1:m_size
      m = m_values[i]
      x = legendre_coef(n,m)
      Approx[:,i] = B*x
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"))
```

```
∴
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
1.0650633160226346
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
0.6086171218702784
```

```
∴
```

```
∴ n = 32; B = legendre(n,M); Approx = zeros(M,m_size-1);
```

```
∴ for i=1:m_size-1
      m = m_values[i+1];
      x = legendre_coef(n,m);
      Approx[:,i] = B*x;
  end
```

```
∴ Plots.plot(T,Approx); display(plot!(T,Y,label="Expected"))
```

```
∴
```

```
∴ for i=1:m_size
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^0.5);
```

```
∴ display(Plots.plot(log.(m_values),err, label="error"))
```

```
∴ num=err[3:m_size] .- err[2:(m_size-1)];
  den=err[2:m_size-1]-err[1:m_size-2]; q=sum(num ./ den)/(m_size-2);
  s=exp(sum(err[2:m_size]-q*err[1:m_size-1])/(m_size-1));
```

```
∴ print("The␣order␣of␣convergence␣is␣estimated␣to␣be","\n",q)
```

```
The order of convergence is estimated to be
0.2408324715806308
```

```
∴ print("The␣rate␣of␣convergence␣is","\n",s)
```

```
The rate of convergence is
3.058988236991122
```

```
∴
```

```
∴ err = zeros(m_size-1);
```

```
∴ for i=1:m_size-1
      err_abs = abs.(Y-Approx[:,i])
      err[i] = dot(err_abs,err_abs)
  end
```

```
∴ err = log.(err .^ 0.5);
```

```
∴ a = [4; 4.5; 5];
```

```
∴ Plots.plot(log.(m_values[1:m_size-1]),err, label="error");
  display(plot!(a, (a*(-1.5) .+ 6), label="slope=-1.5"));
```

```
∴ num=err[3:m_size-1] .- err[2:(m_size-2)];
  den=err[2:m_size-2]-err[1:m_size-3]; q=sum(num ./ den)/(m_size-3);
  s=exp(sum(err[2:m_size-1]-q*err[1:m_size-2])/(m_size-2));
```

```
print("The order of convergence is estimated to be","\n",q)
```

```
The order of convergence is estimated to be
1.396601993488745
```

```
print("The rate of convergence is","\n",s)
```

```
The rate of convergence is
0.46366810375093176
```