Numerics Past Orals Questions

Anne Liu

August 17, 2026

What is QR factorization?

Any square matrix A has a QR decomposition, meaning you can write it as $A = QR$ where Q is an orthogonal matrix and R is upper triangular.

How would you find the QR factorization?

You can find the QR factorization with Gram-Schmidt to compute an orthogonormal set of vectors $\{q_1, \cdots, q_n\}$ that spans the column space of A. The process of Gram-Schmidt is as follows: let

$$v_1 = a_1, \quad q_1 = \frac{v_1}{\|v_1\|}.$$

$$v_2 = a_2 - \langle q_1, a_2 \rangle q_1, \quad q_2 = \frac{v_2}{\|v_2\|}$$

$$\vdots$$

$$v_k = a_k - \sum_{j=1}^{k-1} \langle q_j, a_k \rangle q_j, \quad q_k = \frac{v_k}{\|v_k\|}$$

This is numerically unstable because if, say, q_1 and a_2 are almost parallel then to compute v_2, we have to subtract numbers that are very close together.

Aside: why is subtracting numbers that are close together bad?

Let $f(a, b) = a - b$. Then the absolute condition number of this map is

$$\kappa_{\text{abs}} = \|\nabla f\|$$

and the relative condition number is

$$\kappa_{\text{rel}} = \frac{\|(a, b)\|}{\|f(a, b)\|} \|\nabla f(a, b)\|.$$

In the 1-norm, we have

$$\kappa_{\text{rel}} = \frac{|a| + |b|}{|a - b|} 2$$

and so if $|a - b| << 1$ then this blows up.

Anyways, back to what I was saying... regular Gram-Schmidt is not a very good algorithm and

shouldn't be used in practice for this reason. But there is also modified Gram-Schmidt. This computes each column q_k iteratively as follows.

$$\begin{aligned}
v_j^{(1)} &= a_j \\
v_j^{(2)} &= v_j^{(1)} - \langle q_1, v_j^{(1)} \rangle q_1 \\
v_j^{(3)} &= v_j^{(2)} - \langle q_2, v_j^{(2)} \rangle q_2 \\
&\vdots \\
v_j &= v_j^{(j-1)} - \langle q_{j-1}, v_j^{(j-1)} \rangle q_{j-1} \\
q_j &= \frac{v_j}{\|v_j\|}
\end{aligned}$$

The other way that you can find the QR factorization is through Householder projections. You can think of classical Gram-Schmidt as turning A into an orthogonal matrix by multiplying it on the right by a bunch of upper triangular matrices:

$$A(R_1 R_2 \cdots R_n) = Q$$

where then $R = (R_1 R_2 \cdots R_n)^{-1}$. The Householder reflector method is the opposite idea: multiplying A by orthogonal matrices that will transform it into an upper triangular matrix.

$$(Q_n \cdots Q_1)A = R$$

There are two main types of orthogonal matrices: rotations and reflections. Obviously by the name, the Householder reflectors uses reflection matrices. Reflection matrices take the form

$$\left(I - 2\frac{vv^T}{v^T v}\right)$$

where v is a vector that points orthogonal to the plane of reflection. All three of these algorithms (classic GS, modified GS, Householder) have roughly $\mathbf{2mn^2}$ FLOPs for an $m \times n$ matrix. Thinking about Householder, we are multiplying A by m orthogonal matrices to get R. A matrix-matrix multiplication of two $n \times n$ matrices is $O(n^3)$. I'm actually not sure why this would be $O(mn^2)$ but we didn't cover this in class. Every matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ has a QR factorization. It is unique if we require the diagonal elements of R to be positive.

What's the convergence behavior of conjugate gradient?

The conjugate gradient algorithm can be thought of as steepest descent with book-keeping. Both of these algorithms work when A is **SPD**. We want to solve a linear system $Ax = b$ by minimizing

$$\boxed{f(x) = \frac{1}{2}x^T Ax - b^T x}$$

This of course would solve the system because

$$\nabla f(x) = Ax - b.$$

In the method of steepest descent, we want to start with some initial guess x_0 and then continue to update in the direction of the negative gradient, which happens to be the residual

$$r_k = b - Ax_k.$$

Thus at each step, we update

$$x_{k+1} = x_k + \alpha_k r_k.$$

The choice of α_k that minimizes $f(x_{k+1})$ is

$$\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k}.$$

This algorithm is okay, but the conjugate gradient algorithm leverages all of the past information to converge more quickly. Let $\{p_1, \cdots, p_k\}$ be the previous k steps that we have taken up to this point. There are three things that we want:

1. $\{p_1, \cdots, p_k\}$ are linearly independent.
2. $f(x_k) = \min_{x \in x_0 + \text{span}\{p_1, \cdots, p_k\}} f(x)$
3. x_k can be easily computed from x_{k-1}

Bullet point 1 tells us that we aren't backtracking - we are always making progress. Bullet point 2 tells us that we are doing the best we possibly can at each step. Points 1 and 2 combined tell us that conjugate gradient converges in n iterations! Therefore the CG algorithm lies somewhere between a direct and iterative scheme. We don't want to ever actually run it for n iterations. So we update at each step as follows

$$x_k = x_k + \alpha_k p_k$$

where p_k depends on the previous search vector p_{k-1} and α_k depends on previous things and p_k.

Define orthogonal polynomials and give and example of a family of orthogonal polynomials.

With respect to some $L2$ inner product with some weight function $w(x)$,

$$\langle f, g \rangle_w = \int_a^b f(x) g(x) w(x) dx,$$

we say that two polynomials p_1 and p_2 are orthogonal if

$$\langle p_1, p_2 \rangle_w = 0.$$

The Chebyshev polynomials are orthogonal. They are defined as

$$T_n(\cos\theta) = \cos(n\theta)$$

i.e.

$$T_0(x) = 1,\ T_1(x) = x,\ T_2(x) = 2x^2 - 1, \cdots, T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

These are orthogonal with respect to the weighted inner product

$$\langle f, g \rangle := \int_{-1}^{1} f(x) g(x) \frac{1}{\sqrt{1 - x^2}} dx.$$

How does quadrature relate to orthogonal polynomials?

The general quadrature formula is

$$\hat{I}(f) = \sum_{i=0}^{n} \lambda_i f(x_i).$$

The trapezoidal formula comes from approximating f as piecewise linear. From this you get

$$\hat{I}(f) = \frac{\triangle x}{2} f(x_0) + \triangle x \sum_{i=1}^{n-1} f(x_i) + \frac{\triangle x}{2} f(x_n).$$

The Newton-Cotes formulas come from instead approximating f with polynomials. Suppose we know the values of our function f at nodes $x_0, \cdots, x_n$. Then we can interpolate in the Lagrange basis. Recall that the Lagrange polynomials are the ones such that

$$L_i(x_j) = \delta_{ij}$$

and so the coefficients of the interpolating polynomials are just the values of f at the nodes. Then we have

$$f(x) \approx \sum_{i=0}^{n} f(x_i) L_i(x)$$

and so

$$I(f) \approx \sum_{i=0}^{n} f(x_i) \lambda_i$$

where

$$\lambda_i = \int_a^b L_i(x) dx.$$

The nice thing about these formulas is that the weights only ever have to be computed once since they don't depend on f at all. You can just look them up in a table online. Note that for $n+1$ nodes $x_0, \cdots, x_n$, there exists exactly one quadrature formula (one set of weights λ_i's) that is exact for all polynomials of degree n. In general, the Lagrange polynomials are not orthogonal.

To answer this question, we have to talk about **Gauss-Christoffel quadrature**. The main idea behind Gauss quadrature is that if you can place your nodes judiciously, then you have bought yourself $n+1$ more degrees of freedom, so you should now be able to come up with a quadrature rule that is exact for polynomials up to degree $2n+1$.

Gauss-Christoffel quadrature.

Consider nodes $x_0, \cdots, x_n$ and a quadrature rule

$$\hat{I}(f) = \sum_{i=0}^{n} f(x_i) \lambda_i$$

that is exact for polynomials of degree $2n+1$ for some weighted integral

$$I(f) = \int_a^b f(x) w(x) dx.$$

Then the polynomials $\{P_k\}$ given by

$$P_{k+1} = (x - x_0) \cdots (x - x_k)$$

are orthogonal with respect to the inner product weighted by $w(x)$.

This theorem tells us that the nodes that lead to a quadrature rule that integrates polynomials of degree $2n+1$ exactly must be the roots of orthogonal polynomials with respect to the weighted inner product. We know that for a given weight function, there exists a unique set of orthogonal polynomials (with leading coefficient 1). Thus to summarize...

Summary of Gauss quadrature.

There exists uniquely determined nodes $x_0, \cdots, x_n$ and weights $\lambda_0, \cdots, \lambda_n$ such that the quadrature formula

$$\hat{I}(f) = \sum_{i=0}^{n} \lambda_i f(x_i)$$

integrates exactly all polynomials of degree $2n+1$ with respect to some weight function, i.e.

$$\hat{I}(p) = I(p) = \int_a^b w(x)p(x)dx$$

for all $p \in \mathcal{P}^{2n+1}$. The nodes are the roots of the $(n+1)$st orthogonal polynomial with respect to $w(x)$ and the weights are the Lagrange weights

$$\lambda_i = \int_a^b L_i(x)dx.$$

What is Newton's method?

Newton's method is an iterative method for solving $Ax = b$. The idea is to formulate the problem as a fixed point problem:

$$0 = b - Ax \iff 0 = Q^{-1}(b - Ax) \iff (I - Q^{-1}A)x + Q^{-1}b = x$$

So we are solving some problem

$$F(x) = x$$

where $F(x) = Gx + c$. If

$$\rho(G) < 1$$

then the fixed point iteration $x_{k+1} = F(x_k)$ will converge for any starting point. We get several different iterative schemes by choosing different choices of Q. We want to choose a Q that is close to A but also relatively easy to invert. The two extreme cases would be $Q = I$ (which gives the Richardson method) or $Q = A$ which of course just solves the system directly. Some more happy-medium choices are the Jacobi method

$$Q = D$$

which converges if A is **SDD** and the Newton method

$$Q = L + D$$

which converges if A is **SPD**.

There is something else called Newton's method that is for finding the roots of a nonlinear function, i.e. solving

$$f(x) = 0.$$

The idea behind this method is to choose a starting point x_0, compute the tangent line to f at this point and let x_1 be the root of this line, and continue iteratively. Written out this is

$$x_{k+1} = x_k - \frac{f'(x_k)}{f(x_k)}.$$

For this to converge, we need $f \in C^1$ and $f' \neq 0$.

What is your favorite theorem from Numerics 1?

I'm not sure if this really counts as a theorem but I remember finding the derivation of the convergence speed of the power method for finding eigenvectors to be quite satisfying. I believe the convergence speed is something like

$$O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right)$$

but let's just go through the derivation so I remember. So the power method involves starting with some vector x_0 (and I think the only requirement on x_0 is that it not be orthogonal to the principal eigenvector) and then you successively compute

$$x_{k+1} = \frac{Ax_k}{\|Ax_k\|} = \frac{A^k x_0}{\|A^k x_0\|}.$$

If you assume that A is unitarily diagonalizable, then we can write it as

$$A = V\Lambda V^T$$

and then

$$A^k x_0 = V\Lambda^k V^T x_0.$$

We can write x_0 in terms of the eigenvectors of A since they span $\mathbb{R}^n$:

$$x_0 = \sum_{i=1}^{n} c_i v_i.$$

Then we have

$$\begin{aligned}
A^k x_0 &= V\Lambda^k V^T \sum_{i=1}^{n} c_i v_i \\
&= \sum_{i=1}^{n} c_i V\Lambda^k e_i \\
&= \sum_{i=1}^{n} c_i V\lambda_i^k e_i \\
&= \sum_{i=1}^{n} c_i \lambda_i^k v_i \\
&= c_1\lambda_1^k \left(v_1 + \sum_{i=2}^{n} \frac{c_i}{c_1}\left(\frac{\lambda_i}{\lambda_1}\right)^k v_i\right)
\end{aligned}$$

Therefore we have that

$$x_{k+1} = \frac{A^k x_0}{\|A^k x_0\|} = \frac{v_1 + \sum_{i=2}^n \frac{c_i}{c_1}\left(\frac{\lambda_i}{\lambda_1}\right)^k v_i}{\|v_1 + \sum_{i=2}^n \frac{c_i}{c_1}\left(\frac{\lambda_i}{\lambda_1}\right)^k v_i\|}$$

but since $(\lambda_i/\lambda_1)^k \to 0$ as $k \to \infty$, we have that

$$x_{k+1} \to \frac{v_1}{\|v_1\|}$$

and this convergence behavior is dominated by the rate at which

$$(\lambda_2/\lambda_1)^k \to 0.$$

How do you use gradient descent to solve the linear least squares problem?

The linear least squares problem involves finding

$$\hat{x} = \text{argmin}\frac{1}{2}\|Ax - b\|^2.$$

Let $f(x) = \frac{1}{2}\|Ax - b\|^2$. Then normally people compute

$$\nabla f = \nabla\left(\frac{1}{2}(Ax-b)^T(Ax-b)\right) = A^TAx - A^Tb = 0$$

and obtain the normal equations $A^TAx = A^Tb$. We never solve these directly because the condition number of A^TA is $\kappa(A)^2$:

$$\kappa(A^TA) = \|A^TA\|\|(A^TA)^{-1}\| \leq \|A\|^2\|A^{-1}\|^2 = \kappa(A)^2$$

so what people normally do is use the QR factorization: $A = QR$ because then

$$A^TA = (QR)^T(QR) = R^TQ^TQR = R^TR$$

and the solution is $x = R^{-1}Q^Tb$ if A was square. If $A \in \mathbb{R}^{m\times n}$ where $m > n$ (i.e. the system is overdetermined) then we can use the reduced QR factorization:

$$A = Q_1R_1$$

where $Q_1 \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times n}$. Then our solution is

$$x = R_1^{-1}Q_1^Tb.$$

You could also solve this problem with gradient descent which involves making an initial guess x_0 and then updating in the direction of the negative gradient:

$$x_{k+1} = x_k + \alpha_k(A^Tb - A^TAx_k).$$

Consider the problem $\sin x = rx$ for $r \in (0,1)$. Draw the intersection. How would you solve it?

The problem $\sin x = rx$ can be reformulated as

$$f(x) = \sin x - rx = 0$$

and then we can use Newton's method to solve. To use Newton's method, we need to have $f \in C^1$ which we do, and we need $f' \neq 0$. Let's compute

$$f'(x) = \cos x - r = 0 \implies \cos x = r$$

so actually since $r \in (0, 1)$ this will be satisfied for some x and thus we cannot actually use Newton's method. Another way to solve this would be to think of this as a fixed point problem:

$$\frac{\sin x}{r} = x.$$

We could solve this iteratively by applying the map

$$f : x \mapsto \frac{\sin x}{r}$$

if f is a contraction, i.e. we need to show that there exists some $\theta \in [0, 1)$ such that

$$|f(x) - f(y)| < \theta |x - y|.$$

This boils down to seeing if f is Lipschitz with Lipschitz constant less than 1. Compute

$$f'(x) = \frac{\cos x}{r} > \cos x$$

which can be ≥ 1 so this is not true either.

What is the convergence rate for Newton's method?

First, I want to just restate what Newton's method is. For the n-D case, when we want to solve

$$F(x) = 0$$

for $x \in \mathbb{R}^n$ and $F : \mathbb{R}^n \to \mathbb{R}^n$, the iteration is

$$x_{k+1} = x_k - F'(x_k)^{-1} F(x_k)$$

where $F'(x_k)$ is the Jacobian matrix:

$$\begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \ddots & & \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix}$$

Newton's method converges quadratically, i.e.

$$\|x_{k+1} - x^*\| \leq \frac{w}{2} \|x_k - x^*\|^2$$

to the true solution $x = x^*$ under some conditions on F (including that it is continuously differentiable and F' is invertible) and on the starting position x_0 being sufficiently close to x^*. Because of this second condition, we say that Newton's method converges locally.

Newton's method can be applied to solve the nonlinear least squares problem, which gives rise to the **Gauss-Newton** method. In linear least squares we are computing

$$\min \frac{1}{2}\|Ax - b\|^2.$$

Now, we want to compute

$$\min \frac{1}{2}\|F(x)\|^2.$$

To minimize, we compute the gradient:

$$\begin{aligned}
\nabla \frac{1}{2}\|F(x)\|^2 &= \frac{1}{2}\nabla(F(x)^T F(x)) \\
&= \frac{1}{2}\nabla(F_1(x)\cdots F_n(x)) \begin{pmatrix} F_1(x) \\ \vdots \\ F_n(x) \end{pmatrix} \\
&= \frac{1}{2}\nabla(F_1^2(x) + \cdots + F_n^2(x)) \\
&= \begin{pmatrix} F_1(x)\frac{\partial F_1}{\partial x_1} + \cdots F_n(x)\frac{\partial F_n}{\partial x_1} \\ \vdots \\ F_1(x)\frac{\partial F_1}{\partial x_n} + \cdots F_n(x)\frac{\partial F_n}{\partial x_n} \end{pmatrix} \\
&= F'(x)^T F(x)
\end{aligned}$$

Thus to minimize we want to solve

$$G(x) := F'(x)^T F(x) = 0$$

and so we apply Newton to this. We thus obtain the scheme

$$x_{k+1} = x_k - G'(x_k)^{-1} G(x_k)$$

where we have

$$G'(x) = F'(x)^T F'(x) + F''(x)^T F(x)$$

but we ignore the second term because it's difficult to compute and also if F is small it shouldn't matter much so we use

$$G'(x) = F'(x)^T F'(x).$$

Gauss Newton converges linearly.

What is condition number?

The condition number of a differentiable map $x \mapsto f(x)$ is

$$\kappa_{\text{abs}} = \|f'(x)\|$$

$$\kappa_{\text{rel}} = \frac{\|x\|}{\|f(x)\|}\|f'(x)\|.$$

The condition number tells you how perturbations to the input will change the result, i.e. it gives the bounds

$$\|f(x) - f(y)\| \leq \kappa_{\text{abs}}\|x - y\|$$
$$\frac{\|f(x) - f(y)\|}{\|f(x)\|} \leq \kappa_{\text{rel}}\frac{\|x - y\|}{\|x\|}$$

The relative condition number is the more important quantity and it tells you how a problem is conditioned. If $\kappa_{\text{rel}} \sim 1$, the problem is well-conditioned. If $\kappa_{\text{rel}} >> 1$, the problem is poorly conditioned. If κ_{rel} does not exist, the problem is ill-conditioned.

What is the condition number of multiplication, say $x \mapsto 3x$?

Here, we have $f(x) = 3x$, so

$$\kappa_{\text{abs}} = 3$$

and

$$\kappa_{\text{rel}} = \frac{|x|}{|3x|}3 = 1$$

so multiplication is well-conditioned.

What about for division, i.e. $x \mapsto 3/x$?

$$\kappa_{\text{abs}} = \frac{3}{x^2}$$
$$\kappa_{\text{rel}} = 1$$

so again this is well-conditioned.

How about for addition and subtraction?

Subtracting two numbers that are close together is poorly conditioned. Consider the map

$$f(x, y) = x - y.$$

Then

$$\nabla f(x, y) = [1 \quad -1]$$

and so

$$\kappa_{\text{rel}} = \frac{\|[x \;\; y]\|}{|x - y|}\|[1 \quad -1]\|$$

The 1-norm of a matrix is the maximum column sum, therefore

$$\kappa_{\text{rel}} = \frac{|x| + |y|}{|x - y|}$$

and so if $|x - y| << 1$ then this becomes very large.

Do you know any problems in the fields you have studied that are ill-conditioned?

The canonical example of an ill-conditioned problem from Numerics 1 is that of finding the intersection point between two lines that are close together in slope. *draw picture and explain how a small perturbation in one of the lines leads to a large perturbation in where their intersection point is*

I also know that the Vandermonde matrix which arises from finding the interpolating coefficients in a monomial basis is poorly conditioned. Then they might be like, what is the Vandermonde matrix? And so I would say: if you have $n+1$ nodes $x_0, \cdots, x_n$ and you know $f(x_j)$ then you can find a polynomial of degree n that interpolates)(f at these nodes, i.e. $p(x_j) = f(x_j)$. This polynomial will be of the form

$$p(x) = \sum_{j=0}^{n} c_j x^j$$

and so we want to find the c_j's. To do so we set up a system which just says that

$$p(x_k) = f(x_k)$$

for $k = 0, \cdots, n$, which looks like

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ \vdots & & & & \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} f(x_0) \\ \vdots \\ f(x_n) \end{bmatrix}$$

They might be like why is this matrix poorly conditioned? And I don't exactly know but it's possibly because if the nodes are close together then the rows are close to being linearly dependent and so the matrix is almost not invertible. I would probably say that the condition number of the matrix (in the 2-norm) is

$$\kappa_2(V) = \|V\|_2 \|V^{-1}\|_2 = \frac{\rho_{\max}}{\rho_{\min}}$$

and so it must be the case that $\rho_{\min} << \rho_{\max}$.

List methods for solving linear systems.

I believe that the first method we learned for solving linear systems was the PLU factorization. This is just Gaussian elimination to turn A into an upper triangular matrix, and L stores the weights used to subtract rows. P is a permutation matrix. Once you have $A = PLU$, you can solve $LUx = Pb$ by using forward substitution to solve $Ly = Pb$, then backward substitution to solve $Ux = y$. The computational complexity of backward/forward substitution is $O(n^2)$. The computational complexity of Gaussian elimination is $O(n^3)$, so computing the PLU decomposition is only really useful if you want to solve with many different right hand sides b.

When a linear system is overdetermined, we learned to use the QR factorization for the least-squares problem. The QR factorization can be computed with classic GS, modified GS, or Householder reflections. GS is $O(n^3)$. For an $m \times n$ matrix, the computational complexity of these 3 algorithms is $O(mn^2)$.

Another method for solving linear systems is the conjugate gradient algorithm. This is an iterative method that also converges exactly in n iterations. The idea is to solve $Ax = b$ by minimizing

$$f(x) = \frac{1}{2} x^T A x - b^T x$$

since $\nabla f = Ax - b$. This works if A is SPD because then f is convex. In regular gradient descent we would update in the direction of the negative gradient (which happens to be the residual). The convergence behavior of steepest descent is

$$\|x_k - x^*\|_A \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right)^k \|x_0 - x^*\|_A$$

where $\|x\|_A = \sqrt{x^T Ax}$. Thus if A is well-conditioned and $\kappa_2(A) \approx 1$ then this will converge quickly. For conjugate gradient, we require that all the steps $\{p_1, \cdots, p_k\}$ be linearly independent. To achieve this, we'd like to define our next step to be the residual minus its projection onto the previous steps (via Gram-Schmidt):

$$p_k = r_{k-1} - \sum_{j=1}^{k-1} \frac{\langle r_{k-1}, p_j \rangle_A}{\|p_j\|_A^2} p_j$$

At a high level, you can define steps such that $\langle r_{k-1}, p_j \rangle_A = 0$ for $j = 1, \cdots, k-2$ and thus you don't actually need to store the whole history! So you compute your next step as just

$$p_k = r_{k-1} - \frac{\langle p_{k-1}, r_{k-1} \rangle_A}{\|p_{k-1}\|_A^2} p_{k-1}$$

and similarly the step size only depends on the present information:

$$\alpha_k = \frac{\langle r_{k-1}, p_k \rangle}{\|p_k\|_A^2}$$

Another method for solving linear systems is the Singular Value Decomposition. If A is an $m \times n$ matrix, then we can write

$$A = U \Sigma V^*$$

where $U \in \mathbb{C}^{m \times m}$, $\Sigma \in \mathbb{C}^{m \times n}$, and $V \in \mathbb{C}^{n \times n}$. U and V are unitary matrices and Σ is a diagonal matrix containing the singular values which are the squareroots of the eigenvalues of $A^T A$. The columns of U are the eigenvectors of AA^T and the columns of V are the eigenvectors of $A^T A$. The SVD gives the best rank r approximation of A via

$$A_r := \sum_{i=1}^{r} \sigma_i u_i v_i^*$$

i.e.

$$\|A - A_r\|_2 = \min_{B \text{ is rank r}} \|A - B\|_2$$

The first r columns of U, $\{u_1, \cdots, u_r\}$ span the column space of A, and $\{v_{r+1}, \cdots, v_n\}$ span the kernel of A. The singular values tell us many things about A:

$$\|A\|_2 = \sigma_1$$

$$\|A\|_F = \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}$$

$$\kappa_2(A) = \frac{\sigma_1}{\sigma_r}$$

$$\det(A) = \prod_{i=1}^{r} \sigma_i$$

The nonzero singular values of A are the squareroots of the nonzero eigenvalues of A^*A and AA^*:

$$AA^* = U\Sigma V^* V\Sigma U^* = U\Sigma^2 U^*$$

This is the eigenvalue decomposition of AA^* so its eigenvalues are the entries of Σ^2.

What are the issues with polynomial interpolation using equally spaced points? How can this be alleviated?