

Introduction

In this first lecture we define the problem statement, give an application, recall Newton's method and formulate a statement of the theorem of Bézout.

1 Problem Statement

We consider as given a system $f(\mathbf{x}) = \mathbf{0}$ of N polynomials $f = (f_1, f_2, \dots, f_N)$ in n unknowns $\mathbf{x} = (x_1, x_2, \dots, x_n)$. Depending on whether $N > n$, $N = n$, or $N < n$, we call the system respectively overdetermined, square, and underdetermined. The coefficients of the polynomials are complex numbers and may be known with limited accuracy, as the result of empirical data.

We distinguish three different forms of the input polynomials:

1. The degree is the main characteristic of the polynomials.

The degree is usually the first attribute of a polynomial, we speak of quadratic, cubic, quartic, and quintic polynomials when the degree is respectively 2, 3, 4, and 5.

A polynomial of degree d is dense if all monomials of degree d or less occur with nonzero coefficient. In general, dense polynomials of degree d in n variables have a number of monomials which grows exponentially. Consider Pascal's formula:

$$\binom{d+n}{d} = \binom{d+(n-1)}{d} + \binom{(d-1)+n}{d-1}.$$

The first term of the sum counts the number of monomials of degree d while the second term counts the number of monomials of degree strictly less than d .

As the size of dense polynomials grows exponentially, dense polynomials are not practical to work with.

2. A finite set A of exponent vectors is the support of a polynomial.

Then we denote a polynomial $p(\mathbf{x})$ in multi-index notation as

$$p(\mathbf{x}) = \sum_{\mathbf{a} \in A} c_{\mathbf{a}} \mathbf{x}^{\mathbf{a}}, \quad c_{\mathbf{a}} \in \mathbb{C}, \quad \mathbf{a} = (a_1, a_2, \dots, a_n).$$

For example, $p(x_1, x_2) = c_{12}x_1x_2^2 + c_{20}x_1^2 + c_{11}x_1x_2 + c_{00}$ uses a multi-index notation for the nonzero coefficients. While $\text{degree}(p) = 3$, a quick drawing of all possible third degree monomials in two variables, shows that a dense cubic has 10 coefficients. But our example is sparse: $\#A = 4$.

Most polynomials occurring in practical applications are "sparse", i.e.: only relatively few monomials occur with nonzero coefficient. The sparse structure of a polynomial is then represented by its support or often also by the convex hull of the support, the so-called Newton polygon (or polytope in arbitrary dimensions).

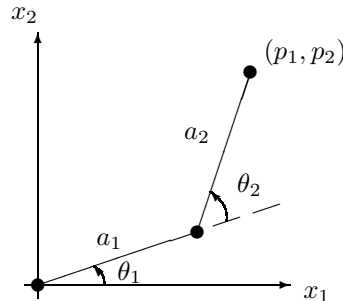
3. A program is given which evaluates the polynomial.

Considering polynomials as functions is done typically in numerical analysis. While evaluating polynomials seems at first sight rather straightforward, finding the optimal way to evaluate a polynomial in several variables is not so obvious. Except for dense polynomials – where the nested Horner scheme is best – the optimal evaluation strategy is often dependent on the type of application.

Since the coefficients of the polynomials are considered to be of limited accuracy, the algorithms we will study will find approximate solutions. In case there are finitely many solutions, we are interested in having a sharp bound on the number of solutions. When there are infinitely many solutions, like surfaces and curves, we are interested in knowing the degrees of the components and the number of irreducible factors.

2 Inverse Kinematics

The design of mechanisms is one application area for polynomial systems. As a first application, consider a planar robot arm, consisting of two links and two joints. The position of the hand of the robot is determined by the angles the two links make. See the figure below.



Given the coordinates (p_1, p_2) for the desired position of the robot hand, the inverse kinematics problem is to determine the values for the angles (θ_1, θ_2) so that the hand reaches the desired position.

To formulate the equations, consider the transformation of the coordinate system so that the new origin is at the first joint, rotated so that the first coordinate axis is aligned with the first link.

$$\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} c_1 & -s_1 & 0 \\ s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotate by } \theta_1} \underbrace{\begin{bmatrix} 1 & 0 & a_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{translate}} \begin{bmatrix} x'_1 \\ x'_2 \\ 1 \end{bmatrix}, \quad s_1 = \sin(\theta_1), c_1 = \cos(\theta_1).$$

If we denote product of the rotation-translation matrices by

$$A_1 = \begin{bmatrix} c_1 & -s_1 & c_1 a_1 \\ s_1 & c_1 & s_1 a_1 \\ 0 & 0 & 1 \end{bmatrix},$$

and similarly A_2 for the coordinate transformation using the second link and the second angle, then we write the two coordinate transformations by the matrix equation

$$\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = A_1 A_2 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} c_1 c_2 a_2 - s_1 s_2 a_2 + c_1 a_1 \\ s_1 c_2 a_2 + c_1 s_2 a_2 + s_1 a_1 \\ 1 \end{bmatrix},$$

which expresses that the hand of the robot has coordinates $(0, 0)$ in the third coordinate system and (x_1, x_2) in the first coordinate system. So the system we must solve is

$$\begin{cases} c_1 c_2 a_2 - s_1 s_2 a_2 + c_1 a_1 = p_1 \\ s_1 c_2 a_2 + c_1 s_2 a_2 + s_1 a_1 = p_2 \\ c_1^2 + s_1^2 = 1 \\ c_2^2 + s_2^2 = 1 \end{cases} \quad (1)$$

with the last two equations respectively expressing $\sin^2(\theta_1) + \cos^2(\theta_1) = 1$ and $\sin^2(\theta_2) + \cos^2(\theta_2) = 1$. With given numbers for the lengths of links a_1 and a_2 and values for the coordinates (p_1, p_2) , we have a system of four quadratic equations to solve the inverse position problem. This example was taken from [6, Chapter 10]. Polynomial systems play a prominent role in the design of linkages [5].

3 Newton's Method

Locally, if we have a good guess about a solution, we may obtain a more accurate solution using Newton's method. The idea for Newton's method has its origin in Taylor series.

For $n = 1$, the Taylor expansion of a function f about x is

$$f(x + \Delta x) = f(x) + \Delta x \frac{f'(x)}{1!} + (\Delta x)^2 \frac{f''(x)}{2!} + (\Delta x)^3 \frac{f'''(x)}{3!} + \dots = \sum_{j=0}^{\infty} (\Delta x)^j \partial_j f(x).$$

To derive Newton's method, we truncate the expansion at the second term:

$$f(x + \Delta x) = f(x) + \Delta x f'(x) + O((\Delta x)^2).$$

If we have that at x : $f(x) \approx 0$, and we look for Δx so that $f(x + \Delta x) = 0$, then we solve the truncated expansion for Δx and obtain $\Delta x = -f(x)/f'(x)$ as the update to the current approximation x . This formula gives rise to a sequence x_k of successive approximations of a root, given by

$$x_{k+1} := x_k - \frac{f(x_k)}{f'(x_k)}, \quad \text{for } k = 0, 1, \dots$$

which is well defined provided the derivative f' does not vanish. If f' does not vanish at the root, then the sequence converges quadratically for x_0 close enough to the root, i.e.: if $f(x^*) = 0$: $\|x_{k+1} - x^*\| \leq \|x_k - x^*\|^2$. We can formalize the algorithm using MATLAB (or Octave):

```
function [x, fail] = newton(f, df, x, eps, N)
%
% Applies Newton's method to the function f, with derivative in df,
% starting at the point x. It produces a sequence of approximations.
% Stops when the sequence reaches the point x where
% |f(x)| < eps or when two consecutive points in the sequence
% have distance less than eps apart from each other.
% Failure is reported (fail = 1) when the accuracy requirement
% is not satisfied in N steps; otherwise fail = 0 on return.
%
% Example :
% >> [x, fail] = newton('sin', 'cos', pi/4, 1e-8, 10)
%
fprintf('running the method of Newton...\n');
fx = feval(f, x);
for i = 1:N
    dx = fx/feval(df, x);
    x = x - dx;
    fx = feval(f, x);
    fprintf(' x = %e, dx = %e, f(x) = %e\n', x, dx, fx);
    if (abs(fx) < eps) | (abs(dx) < eps)
        fail = 0;
        fprintf('succeeded after %d steps\n', i);
        return;
    end
end
fprintf('failed requirements after %d steps\n', N);
fail = 1;
```

For $n = 2$, we generalize the Taylor expansion of f as

$$f(x_1 + \Delta x_1, x_2 + \Delta x_2) = f(x_1, x_2) + \frac{\Delta x_1}{1!} \frac{\partial f}{\partial x_1}(x_1, x_2) + \frac{\Delta x_2}{1!} \frac{\partial f}{\partial x_2}(x_1, x_2) \\ + \frac{(\Delta x_1)^2}{2!} \frac{\partial^2 f}{\partial x_1^2}(x_1, x_2) + \frac{(\Delta x_1)(\Delta x_2)}{1!1!} \frac{\partial^2 f}{\partial x_1 \partial x_2}(x_1, x_2) + \frac{(\Delta x_2)^2}{2!} \frac{\partial^2 f}{\partial x_2^2}(x_1, x_2) + \cdots = \sum_{|\mathbf{a}|=0}^{\infty} (\Delta x)^{\mathbf{a}} \partial_{\mathbf{a}} f(x),$$

where for $\mathbf{a} = (a_1, a_2, \dots, a_n)$ the operator $\partial_{\mathbf{a}}$ is defined by

$$\partial_{\mathbf{a}} = \frac{1}{a_1! a_2! \cdots a_n!} \frac{\partial^{|\mathbf{a}|}}{\partial_1^{a_1} \partial_2^{a_2} \cdots \partial_n^{a_n}}, \quad \text{with } |\mathbf{a}| = a_1 + a_2 + \cdots + a_n.$$

If a function is differentiable, then it means that locally we may replace the function by a linear function. For $\Delta x \approx 0$, $f(x) \approx$ linear. To derive Newton's method in several variables, we write the Taylor expansions in matrix form:

$$f(\mathbf{x} + \Delta \mathbf{x}) = f(\mathbf{x}) + \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_N}{\partial x_2} & \cdots & \frac{\partial f_N}{\partial x_n} \end{bmatrix} (\Delta \mathbf{x}) + O((\Delta \mathbf{x})^2).$$

The matrix which appears in this expansion is the Jacobian matrix, which we denote by J_f . Let $N = n$, so f is a square system.

Assume that x is a current approximation for the root \mathbf{x}^* (for which $f(\mathbf{x}^*) = 0$) and that we wish to find $\Delta \mathbf{x}$ so that $\mathbf{x} + \Delta \mathbf{x} = \mathbf{x}^*$. Ignoring the second order terms in the Taylor expansion of $f(\mathbf{x} + \Delta \mathbf{x}) = 0$, we have $f(\mathbf{x}) + J_f \Delta \mathbf{x} = 0$. If J_f is invertible, we may solve for $\Delta \mathbf{x}$ as solution of the linear system $J_f \Delta \mathbf{x} = -f(\mathbf{x})$ which gives the update which brings us closer to the root \mathbf{x}^* .

We say that \mathbf{x}^* is a regular root if $\det(J_f(\mathbf{x}^*)) \neq 0$. Otherwise, \mathbf{x}^* is a singular root. Singular roots occur at multiple solutions and solutions that are nonisolated.

A solution \mathbf{x}^* is isolated if there exists a ball centered at \mathbf{x}^* with a nonzero radius that contains no other solution of the system. When a solution is not isolated it lies on a curve or surface. Note that real solutions may occur as isolated real points on a complex curve, i.e.: there is no other real solution in its neighborhood. Consider for example the single equation $x^2 + y^2 = 0$ defining a pair of complex conjugated lines $y = \pm \sqrt{-1}x$ through the point $(0, 0)$. The point $(0, 0)$ is just as nonisolated as all the other solutions satisfying $x^2 + y^2 = 0$, but it is the only real solution and thus one could regard it as an isolated *real* solution.

Newton's method is one of the most important algorithms to solve nonlinear problems, see [7] for an introduction and [3] for an advanced treatment.

4 The Theorem of Bézout

For general nonlinear systems, we cannot make a statement about the number of isolated solutions. For polynomial systems, the main theorem of algebraic geometry is the theorem of Bézout. We can view this theorem as a generalization [4] of the fundamental theorem of algebra:

Theorem 4.1 (the fundamental theorem of algebra) *A polynomial of degree d in one variable with nonzero leading coefficient has exactly d complex roots, where each root is counted with its multiplicity.*

For a univariate polynomial p , the multiplicity of z as a root of p is k if in addition to $p(z) = 0$ also all derivatives of p up to order $k - 1$ vanish at z .

If we consider a system of two univariate polynomials, of degrees d_1 and d_2 respectively, then it is obvious that there are $d_1 \times d_2$ isolated solutions. The theorem of Bézout says that a system can have no more isolated solutions than the product of the degrees of the polynomials. Formally we can write

Theorem 4.2 (theorem of Bézout on #isolated solutions) Consider a system $f(\mathbf{x}) = \mathbf{0}$ of n equations $f = (f_1, f_2, \dots, f_n)$ in n unknowns $\mathbf{x} = (x_1, x_2, \dots, x_n)$. The number of isolated solutions in \mathbb{C}^n is bounded by

$$D = \prod_{i=1}^n \deg(f_i).$$

We call D the total degree of the system. Also here the solutions have to be counted with multiplicities, although a precise definition of the multiplicity is much harder for polynomial systems. The total degree is a somewhat crude but important complexity bound [2] for polynomial systems.

One proof of the theorem of Bézout works by deforming the polynomials in the system into polynomials in one variable. This kind of proof by deformation leads to homotopy continuation methods for solving polynomial systems. There is a recent translation [1] of Bézout's original monograph.

5 Exercises

1. The notion of a sparse polynomial changes when one considers the number of vertices which span its Newton polygon, i.e.: a dense polynomial may have a Newton polygon which is spanned by only a few vertices. Give an example of two polynomials which have the same Newton polygon where one polynomial is sparse and the other is dense.
2. How many arithmetical operations are needed to evaluate a polynomial of degree d in n variables with a Horner scheme? Start with $n = 1$ and generalize.
3. How many solutions do you expect to the inverse position problem for the planar robot arm with two links? Make a case analysis.
4. Use Maple to solve the inverse position problem for the planar robot arm with two links? In your answer, give the Maple instructions you used and report on the results Maple returns to you. Open source alternatives to Maple, e.g.: CoCoA, Macaulay 2, Sage, or Singular may be used as well.
5. Generalize the MATLAB script `newton` so that it also works for systems. You can of course also use Octave. Give a numerical example to illustrate that your generalized `newton` script works.
6. Apply the theorem of Bézout to bound the number of isolated solutions to the system which must be solved for the inverse position problem for the planar robot arm with two links. Can you derive an equivalent formulation for this problem which leads to a system with lower total degree? (Hint: consider multiplying the matrix equation by A_1^{-1} .)
7. Use Maple or Sage (free to download from www.sagemath.org) and make a worksheet or notebook to derive the polynomial equations in (1) for the inverse kinematics problem. Also derive the alternative formulation suggested in the previous exercise, multiplying with A^{-1} .

References

- [1] E. Bézout. *General Theory of Algebraic Equations*. Princeton University Press, 2006. Translated by Eric Feron. Original French edition published as *Théorie générale des équations algébriques* by Ph.-D. Pierres, Paris, 1779.
- [2] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.
- [3] P. Deuffhard. *Newton Methods for Nonlinear Problems. Affine Invariance and Adaptive Algorithms*. Springer-Verlag, 2004.

- [4] T.Y. Li. Solving polynomial systems. *The Mathematical Intelligencer*, 9(3):33–39, 1987.
- [5] J.M. McCarthy. *Geometric Design of Linkages*. Springer-Verlag, 2000.
- [6] A. Morgan. *Solving polynomial systems using continuation for engineering and scientific problems*. Prentice-Hall, 1987.
- [7] T. Sauer. *Numerical Analysis*. Pearson Addison-Wesley, 2006.