Chapter IV

Linear Algebra Applications

Numerical linear algebra underlies many numerical methods in applications, from simulating fluids, to understanding data and neural networks. Here we briefly investigate some applications, allowing us to go beyond our preliminary numerical algorithms from Chapter I, giving methods for approximating functions, a more powerful numerical method for computing integrals, and the ability to solve ordinary differential equations.

1. IV.1 Polynomial Interpolation and Regression: Often in data science one needs to approximate data by a polynomial. We discuss polynomial interpolation and see how it can be used to compute integrals. We also discuss regression, where more data is used than the degree of the polynomial, leading to a robust approach produced by solving a rectangular least squares problem.

2. IV.2 Differential Equations: Divided differences can can be used to *discretise* differential equations. That is, we can recast differential equation as (approximate) solutions to linear systems. We investigate using this approach on some simple linear initial and boundary value problems.

IV.1 Polynomial Interpolation and Regression

Polynomial interpolation is the process of finding a polynomial that equals data at a precise set of points. In this section we see how an interpolant can be constructed by either solving a linear system involving the Vandermonde matrix, or directly in terms of the Lagrange basis for polynomials. We also investigate an application of polynomial interpolation to computing integrals, giving an alternative to the rectangular and triangular rules from the first chapter. In the lab we see that this leads to much more accurate computation. We also see in the lab that polynomial interpolation has issues, particular with an evenly spaced grid or with a monomial basis. Overcoming this will motivate orthogonal polynomials later in the module.

A more robust scheme that overcomes some of the issues with naive polynomial interpolation is *polynomial regression*, where we use more data than the degrees of freedom in the polynomial. We can determine such a polynomial by solving a *least squares peroblem*: instead of insisting that the polynomial matches data exactly, we find the polynomial whose samples at the points are as close as possible to the data, as measured in the 2-norm.

IV.1.1 Polynomial interpolation

Our prelimary goal is given a set of points and data at those points, usually samples of a function $f_j = f(x_j)$, find a polynomial that interpolates the data at the points:

Definition 25 (interpolatory polynomial)**.** Given *distinct* points $\boldsymbol{x} = [x_1, \dots, x_n]^\top \in \mathbb{C}^n$ and *data* $\boldsymbol{f} = [f_1, \dots, f_n]^\top \in \mathbb{C}^n$, a degree $n-1$ *interpolatory polynomial* $p(x)$ satisfies

$$p(x_j) = f_j$$

The easiest way to solve this problem is to invert the Vandermonde system:

Definition 26 (Vandermonde)**.** The *Vandermonde matrix* associated with $\boldsymbol{x} \in \mathbb{C}^m$ is the matrix

$$V_{\boldsymbol{x},n} := \begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & \cdots & x_m^{n-1} \end{bmatrix} \in \mathbb{C}^{m \times n}.$$

When it is clear from context we omit the subscripts $\boldsymbol{x}, n$.

Writing the coefficients of a polynomial

$$p(x) = \sum_{k=0}^{n-1} c_k x^k$$

as a vector $\boldsymbol{c} = [c_0, \dots, c_{n-1}]^\top \in \mathbb{C}^n$, we note that V encodes the linear map from coefficients to values at a grid, that is,

$$V\boldsymbol{c} = \begin{bmatrix} c_0 + c_1 x_1 + \cdots + c_{n-1} x_1^{n-1} \\ \vdots \\ c_0 + c_1 x_m + \cdots + c_{n-1} x_m^{n-1} \end{bmatrix} = \begin{bmatrix} p(x_1) \\ \vdots \\ p(x_m) \end{bmatrix}.$$

In the square case (where $m = n$), the coefficients of an interpolatory polynomial are given by $\boldsymbol{c} = V^{-1}\boldsymbol{f}$, so that

$$\begin{bmatrix} p(x_1) \\ \vdots \\ p(x_n) \end{bmatrix} = V\boldsymbol{c} = VV^{-1}\boldsymbol{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}.$$

This inversion is justified by the following:

Proposition 11 (interpolatory polynomial uniqueness)**.** *Interpolatory polynomials are unique and therefore square Vandermonde matrices are invertible.*

Proof Suppose p and $\tilde{p}$ are both interpolatory polynomials of the same function. Then $p(x) - \tilde{p}(x)$ vanishes at n distinct points x_j. By the fundamental theorem of algebra it must be zero, i.e., $p = \tilde{p}$.

For the second part, if $V\boldsymbol{c} = 0$ for $\boldsymbol{c} = [c_0, \dots, c_{n-1}]^\top \in \mathbb{C}^n$ then for $q(x) = c_0 + \cdots + c_{n-1}x^{n-1}$ we have

$$q(x_j) = \boldsymbol{e}_j^\top V\boldsymbol{c} = 0$$

hence q vanishes at n distinct points and is therefore 0, i.e., $\boldsymbol{c} = 0$.

■

We can invert square Vandermonde matrix numerically in $O(n^3)$ operations using the PLU factorisation. But it turns out we can also construct the interpolatory polynomial directly, and evaluate the polynomial in only $O(n^2)$ operations. We will use the following polynomials which equal 1 at one grid point and zero at the others:

Definition 27 (Lagrange basis polynomial)**.** The *Lagrange basis polynomial* is defined as

$$\ell_k(x) := \prod_{j\neq k} \frac{x - x_j}{x_k - x_j} = \frac{(x - x_1)\cdots(x - x_{k-1})(x - x_{k+1})\cdots(x - x_n)}{(x_k - x_1)\cdots(x_k - x_{k-1})(x_k - x_{k+1})\cdots(x_k - x_n)}$$

Plugging in the grid points verifies that $\ell_k(x_j) = \delta_{kj}$.

We can use the Lagrange basis to directly construct the interpolatory polynomial:

Theorem 9 (Lagrange interpolation)**.** *The unique interpolation polynomial is:*

$$p(x) = f_1\ell_1(x) + \cdots + f_n\ell_n(x)$$

Proof It follows from inspection:

$$p(x_j) = \sum_{k=1}^{n} f_k\ell_k(x_j) = f_j.$$

■

Example 18 (interpolating an exponential)**.** We can interpolate $\exp(x)$ at the points $0, 1, 2$. That is, our data is $\boldsymbol{f} = [1, \mathrm{e}, \mathrm{e}^2]^\top$ and the interpolatory polynomial is

$$\begin{aligned} p(x) &= \ell_1(x) + \mathrm{e}\ell_2(x) + \mathrm{e}^2\ell_3(x) = \frac{(x-1)(x-2)}{(-1)(-2)} + \mathrm{e}\frac{x(x-2)}{(-1)} + \mathrm{e}^2\frac{x(x-1)}{2} \\ &= (1/2 - \mathrm{e} + \mathrm{e}^2/2)x^2 + (-3/2 + 2\mathrm{e} - \mathrm{e}^2/2)x + 1 \end{aligned}$$

Remark Interpolating at evenly spaced points is a really *bad* idea as it is inherently ill-conditioned. The lab explores this issue experimentally. Another serious issue is that monomials are a horrible basis for interpolation. This is intuitive: when n is large x^n is basically zero near the origin and hence x_j^n numerically lose linear independence, that is, on a computer they appear to be linearly dependent (up to rounding errors). Use alternative sets of points and bases entirely overcomes this issue.

IV.1.2 Interpolatory quadrature rules

Interpolation leads naturally to quadrature rules where one integrates the interpolatory polynomial exactly. This can be viewed as an extension of one-panel Rectangular Rules (which are degree 0 interpolants at a single point) and Trapezium Rules (which are degree 1 interpolants at two points). Using the Lagrange basis for interpolation we can write general interpolatory quadrature rules as a simple weighted sum:

Definition 28 (interpolatory quadrature rule)**.** Given a set of points $\boldsymbol{x} = [x_1, \ldots, x_n]^\top$ the interpolatory quadrature rule is:

$$\Sigma_n^{w,\boldsymbol{x}}[f] := \sum_{j=1}^{n} w_j f(x_j)$$

where

$$w_j := \int_a^b \ell_j(x)w(x)\mathrm{d}x.$$

The convergence of such a scheme is explored in the lab. But an important feature is that it is exact for all low degree polynomials:

Proposition 12 (interpolatory quadrature is exact for polynomials)**.** *Interpolatory quadrature is exact for all degree $n-1$ polynomials p:*

$$\int_a^b p(x)w(x)\mathrm{d}x = \Sigma_n^{w,\boldsymbol{x}}[p]$$

Proof The result follows since, by uniqueness of interpolatory polynomial, if p is a polynomial then

$$p(x) = \sum_{j=1}^n p(x_j)\ell_j(x)$$

Hence

$$\int_a^b p(x)w(x)\mathrm{d}x = \sum_{j=1}^n p(x_j)\int_a^b \ell_j(x)w(x)\mathrm{d}x = \Sigma_n^{w,\boldsymbol{x}}[p].$$

■

Example 19 (3-point interpolatory quadrature)**.** We find the interpolatory quadrature rule for $w(x) = 1$ on $[0,1]$ with points $[x_1, x_2, x_3] = [0, 1/4, 1]$. We have:

$$\begin{aligned} w_1 &= \int_0^1 w(x)\ell_1(x)\mathrm{d}x = \int_0^1 \frac{(x-1/4)(x-1)}{(-1/4)(-1)}\mathrm{d}x = -1/6 \\ w_2 &= \int_0^1 w(x)\ell_2(x)\mathrm{d}x = \int_0^1 \frac{x(x-1)}{(1/4)(-3/4)}\mathrm{d}x = 8/9 \\ w_3 &= \int_0^1 w(x)\ell_3(x)\mathrm{d}x = \int_0^1 \frac{x(x-1/4)}{3/4}\mathrm{d}x = 5/18 \end{aligned}$$

That is we have

$$\Sigma_n^{w,\boldsymbol{x}}[f] = -\frac{f(0)}{6} + \frac{8f(1/4)}{9} + \frac{5f(1)}{18}.$$

This is indeed exact for polynomials up to degree 2 (and no more):

$$\Sigma_n^{w,\boldsymbol{x}}[1] = 1, \Sigma_n^{w,\boldsymbol{x}}[x] = 1/2, \Sigma_n^{w,\boldsymbol{x}}[x^2] = 1/3, \Sigma_n^{w,\boldsymbol{x}}[x^3] = 7/24 \neq 1/4.$$

IV.1.3 Polynomial regression

In many settings interpolation is not an accurate or appropriate tool. Data is often on an evenly spaced grid in which case (as seen in the lab) interpolation breaks down catastrophically. Or the data is noisy and one ends up over resolving: approximating the noise rather than the signal. A simple solution is *polynomial regression*: use more sample points than the degrees of freedom in the polynomial. The special case of an affine polynomial is called *linear regression*.

More precisely, for $\boldsymbol{x} \in \mathbb{C}^m$ and for $n < m$ we want to find a degree $n-1$ polynomial

$$p(x) = \sum_{k=0}^{n-1} c_k x^k$$

such that

$$\begin{bmatrix} p(x_1) \\ \vdots \\ p(x_m) \end{bmatrix} \approx \underbrace{\begin{bmatrix} f_1 \\ \vdots \\ f_m \end{bmatrix}}_{\boldsymbol{f}}.$$

Mapping between coefficients $\boldsymbol{c} \in \mathbb{C}^n$ to polynomial values on a grid can be accomplished via rectangular Vandermonde matrices. In particular, our goal is to choose $\boldsymbol{c} \in \mathbb{C}^n$ so that

$$V\boldsymbol{c} = \begin{bmatrix} p(x_1) \\ \vdots \\ p(x_m) \end{bmatrix} \approx \boldsymbol{f}.$$

We do so by solving the *least squares* system: given $V \in \mathbb{C}^{m \times n}$ and $\boldsymbol{f} \in \mathbb{C}^m$ we want to find $\boldsymbol{c} \in \mathbb{C}^n$ such that

$$\|V\boldsymbol{c} - \boldsymbol{f}\|$$

is minimal. Note interpolation is a special case where this norm is precisely zero (which is indeed minimal), but in general this norm may be rather large. We will discuss the numerical solution of least squares problems in the next few sections.

Remark Using regression instead of interpolation can overcome the issues with evenly spaced grids. However, the monomial basis is still very problematic.

IV.2 Differential Equations via Finite Differences

Linear algebra is a powerful tool for solving linear equations, including ∞-dimensional ones like differential equations. In this section we discuss *finite differences*: an algorithmic way of reducing ODEs to linear systems by replacing derivatives with divided difference approximations.

We will focus on the following differential equations. Indefinite integration for $a \leq x \leq b$ can be viewed as solving a very simple first-order linear ODE: given a constant $c \in \mathbb{F}$ (where $\mathbb{F}$ is either $\mathbb{R}$ or $\mathbb{C}$) and a function $f : [a, b] \to \mathbb{F}$, find a differentiable function $u : [a, b] \to \mathbb{F}$ such that

$$\begin{aligned} u(a) &= c, \\ u'(x) &= f(x). \end{aligned}$$

We will then allow for more complicated first order linear ODEs with variable coefficients: given a constant c and functions $f, \omega : [a, b] \to \mathbb{F}$ find u such that

$$\begin{aligned} u(a) &= c, \\ u'(x) - \omega(x)u(x) &= f(x). \end{aligned}$$

For second-order differential equations you may have seen *initial value problems* where the value and derivative at an initial point $x = a$ are provided. Instead, we will consider *boundary value problems* where the value at the left and right endpoints are imposed. In particular we will consider the Poisson equation with *Dirichlet conditions* (i.e. conditions on the left and

right of an interval): given constants c, d and a function f find a twice-differentiable function u such that

$$\begin{aligned} u(a) &= c, \\ u''(x) &= f(x), \\ u(b) &= d \end{aligned}$$

In higher dimensions, the Poisson equation (and other *elliptic* partial differential equations) typically have boundary conditions imposed on the boundary of a complicated geometry, and give the temperature equilibrium of a plate where the temperature is held fixed. The techniques we discuss for our simple 1D model problem extend to these more challenging settings.

Briefly, the basic idea of finite differences is a systematic way of reducing a differential equation to a linear system. For each problem we will do the following steps:

1. Discretise the interval $[a, b]$ by a grid of points $x_0, \ldots, x_n$ and write the ODE on each grid point.
2. Replace true derivatives of the solution u with approximate derivatives in terms of values on the grid $u(x_j)$ via the divided difference formula.
3. In the formula replace unknown values of $u(x_j)$ at the grid by new unknowns u_j.
4. Deduce from this a linear system that can be solved to compute u_j so that (hopefully) $u_j \approx u(x_j)$.

Remark There is a rich theory proving convergence of finite difference approximations to the true solution of ODEs but this is beyond the scope of this module, instead, we just learn the recipe. In the lab we determine convergence rates experimentally.

IV.2.1 Indefinite integration

We begin with the simplest differential equation on an interval $[a, b]$:

$$\begin{aligned} u(a) &= c, \\ u'(x) &= f(x) \end{aligned}$$

As in integration we will use an evenly spaced grid $a = x_0 < x_1 < \ldots < x_n = b$ defined by $x_j := a + hj$ where $h := (b - a)/n$. The solution is of course $u(x) = c + \int_a^x f(x)\mathrm{d}x$ and we could use Rectangular or Trapezium rules to obtain approximations to $u(x_j)$ for each j, however, we shall take another approach that will generalise to other differential equations.

Consider a divided difference approximation like right-sided divided differences:

$$u'(x) \approx \frac{u(x + h) - u(x)}{h}.$$

When applied to a grid point $x_j \in \{x_0, \ldots, x_{n-1}\}$ this becomes:

$$u'(x_j) \approx \frac{u(x_j + h) - u(x_j)}{h} = \frac{u(x_{j+1}) - u(x_j)}{h}$$

Note that x_n is not permitted since that would depend on $u(x_{n+1})$, but $x_{n+1} > b$ and we have only assumed f is defined on $[a, b]$. We use this approximation as follows:

(1) Write the ODE and initial conditions on the grid. Since right-sided divided differences will depend on x_j and x_{j+1} we stop at x_{n-1} to avoid going past our grid:

$$\begin{bmatrix} u(x_0) \\ u'(x_0) \\ u'(x_1) \\ \vdots \\ u'(x_{n-1}) \end{bmatrix} = \underbrace{\begin{bmatrix} c \\ f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n-1}) \end{bmatrix}}_{\boldsymbol{b}}$$

(2) Replace derivatives with divided differences, changing equality to an approximation:

$$\begin{bmatrix} u(x_0) \\ (u(x_1) - u(x_0))/h \\ (u(x_2) - u(x_1))/h \\ \vdots \\ (u(x_n) - u(x_{n-1})/h \end{bmatrix} \approx \boldsymbol{b}$$

(3) We do not know $u(x_j)$ hence we replace it with other unknowns u_j, but where the approximation is turned back into an equality: we want to find $u_0, \ldots, u_n$ such that

$$\begin{bmatrix} u_0 \\ (u_1 - u_0)/h \\ (u_2 - u_1)/h \\ \vdots \\ (u_n - u_{n-1})/h \end{bmatrix} = \boldsymbol{b}$$

(4) This can be rewritten as a lower bidiagonal linear system:

$$\underbrace{\begin{bmatrix} 1 & & & \\ -1/h & 1/h & & \\ & \ddots & \ddots & \\ & & -1/h & 1/h \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{bmatrix}}_{\boldsymbol{u}} = \boldsymbol{b}$$

We can determine $\boldsymbol{u}$ by solving $L\boldsymbol{u} = \boldsymbol{b}$ using forward-substitution.

As mentioned before, it is possible to prove that as $n \to \infty$ we have for all j that $u_j \to u(x_j)$ (uniformly). But we will leave this to be shown experimentally in the lab.

IV.2.2 Forward Euler

We can extend this to more general first-order linear differential equations with a variable coefficient:

$$u(a) = c$$
$$u'(x) + \omega(x)u(x) = f(x)$$

The steps proceed very similar to before:

(1) Write the ODE and initial conditions on the grid, avoiding x_n so that we don't go past the endpoint:

$$\begin{bmatrix} u(x_0) \\ u'(x_0)+\omega(x_0)u(x_0) \\ u'(x_1)+\omega(x_1)u(x_1) \\ \vdots \\ u'(x_{n-1})+\omega(x_{n-1})u(x_{n-1}) \end{bmatrix} = \underbrace{\begin{bmatrix} c \\ f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{n-1}) \end{bmatrix}}_{\boldsymbol{b}}$$

(2) Replace derivatives with divided differences:

$$\begin{bmatrix} u(x_0) \\ (u(x_1)-u(x_0))/h+\omega(x_0)u(x_0) \\ (u(x_2)-u(x_1))/h+\omega(x_1)u(x_1) \\ \vdots \\ (u(x_n)-u(x_{n-1}))/h+\omega(x_{n-1})u(x_{n-1}) \end{bmatrix} \approx \boldsymbol{b}$$

(3) Replace $u(x_j)$ by its approximation u_j but now with the system being an equality:

$$\begin{bmatrix} u_0 \\ (u_1-u_0)/h+\omega(x_0)u_0 \\ (u_2-u_1)/h+\omega(x_1)u_1 \\ \vdots \\ (u_n-u_{n-1})/h+\omega(x_{n-1})u_{n-1} \end{bmatrix} = \boldsymbol{b}$$

(4) This is equivalent to the linear system:

$$\underbrace{\begin{bmatrix} 1 & & & \\ \omega(x_0)-1/h & 1/h & & \\ & \ddots & \ddots & \\ & & \omega(x_{n-1})-1/h & 1/h \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{bmatrix}}_{\boldsymbol{u}} = \boldsymbol{b}$$

We can solve $L\boldsymbol{u} = \boldsymbol{b}$ using forward-substitution so that $u_j \approx u(x_j)$.

IV.2.3 Poisson equation

Consider the Poisson equation with Dirichlet conditions (a two-point boundary value problem): given constants c, d and a function $f : [a, b] \to \mathbb{R}$ find a twice differentiable function $u : [a, b] \to \mathbb{R}$ satisfying

$$\begin{aligned} u(a) &= c, \\ u''(x) &= f(x), \\ u(b) &= d \end{aligned}$$

We shall adapt the procedure using the second-order divided difference approximation from the first probem sheet:

$$u''(x) \approx \frac{u(x-h)-2u(x)+u(x+h)}{h^2}$$

When applied to a grid point $x_1, \ldots, x_{n-1}$ this becomes:

$$u'(x_j) \approx \frac{u(x_j - h) - 2u(x_j) + u(x_j + h)}{h^2} = \frac{u(x_{j-1}) - 2u(x_j) + u(x_{j+1})}{h^2}$$

Note that x_0 and x_n are not permitted since that would go past the endpoints of the interval. We use this approximation as follows:

(1) Write the ODE and boundary conditions on the grid (putting the left condition on the top and right condition on the bottom):

$$\begin{bmatrix} u(x_0) \\ u''(x_1) \\ u''(x_2) \\ \vdots \\ u''(x_{n-1}) \\ u(x_n) \end{bmatrix} = \underbrace{\begin{bmatrix} c \\ f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ d \end{bmatrix}}_{\boldsymbol{b}}$$

(2) Replace derivatives with divided differences:

$$\begin{bmatrix} u(x_0) \\ \frac{u(x_0)-2u(x_1)+u(x_2)}{h^2} \\ \frac{u(x_1)-2u(x_2)+u(x_3)}{h^2} \\ \vdots \\ \frac{u(x_{n-2})-2u(x_{n-1})+u(x_n)}{h^2} \\ u(x_n) \end{bmatrix} \approx \boldsymbol{b}$$

(3) Replace $u(x_j)$ by its approximation u_j: we want to find $u_0, \ldots, n_n$ so that

$$\begin{bmatrix} u_0 \\ \frac{u_0-2u_1+u_2}{h^2} \\ \frac{u_1-2u_2+u_3}{h^2} \\ \vdots \\ \frac{u_{n-2}-2u_{n-1}+u_n}{h^2} \\ u_n \end{bmatrix} = \boldsymbol{b}$$

(4) This is equivalent to a tridiagonal linear system:

$$\underbrace{\begin{bmatrix} 1 & & & & \\ 1/h^2 & -2/h^2 & 1/h^2 & & \\ & \ddots & \ddots & \ddots & \\ & & 1/h^2 & -2/h^2 & 1/h^2 \\ & & & & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_n \end{bmatrix}}_{\boldsymbol{u}} = \boldsymbol{b}$$

This can be solved in $O(n)$ complexity using a banded LU or QR factorisation.