ChatGPT

Clamped Cubic Splines

A **clamped cubic spline** is a piecewise cubic interpolant $S(x)$ passing through given data points $x_0< x_1<\dots< x_n$ such that $S(x_i)=f_i$ and additionally the end slopes are specified:
$$S'(x_0)=f'(x_0),\quad S'(x_n)=f'(x_n)\,. $$
These *clamped boundary conditions* fix the otherwise free end parameters. On each interval $[x_i,x_{i+1}]$, $S(x)$ is a cubic polynomial $$S_i(x)=a_i + b_i (x-x_i) + c_i (x-x_i)^2 + d_i (x-x_i)^3,$$ with coefficients chosen so that $S_i(x_i)=f_i$, $S_i(x_{i+1})=f_{i+1}$, and $S_i'(x_{i+1})=S_{i+1}'(x_{i+1})$, $S_i''(x_{i+1})=S_{i+1}''(x_{i+1})$ for $i=0,\dots,n-2$, plus the clamped end slopes at x_0 and x_n. Equivalently, one introduces $M_i=S''(x_i)$ and derives the well-known compact formula (e.g. by integrating a linear second-derivative):

> **Spline formula (Chasnov [30]):** On $[x_i,x_{i+1}]$ with $h_i=x_{i+1}-x_i$,
> $$S_i(x)\;=\;\frac{(x_{i+1}-x)^3}{6h_i}\,M_i+\frac{(x-x_i)^3}{6h_i}\,M_{i+1} + \frac{x_{i+1}-x}{h_i} \Bigl(f_i-\frac{h_i^2}{6}M_i\Bigr) \;+\; \frac{x-x_i}{h_i}\Bigl(f_{i+1}-\frac{h_i^2}{6}M_{i+1}\Bigr)\,. $$
> Here $M_i=S''(x_i)$ are the (unknown) nodal second derivatives [1] [2].

Requiring $S'(x)$ be continuous at interior nodes x_i yields for $i=1,\dots,n-1$ the linear equations (by equating $S'*{i-1}(x_i)=S'_i(x_i)$: $$h$$ This tridiagonal system comes from (for example) Chasnov's derivation [3] [4]. The clamped end conditions supply the two additional equations. From the first interval formula one finds (setting $x=x_0$ in $S'} M_{i-1} + 2(h_{i-1}+h_i) M_i + h_i M_{i+1} \;=\; 6\Bigl[\frac{f_{i+1}-f_i}{h_i} - \frac{f_i-f_{i-1}}{h_{i-1}}\Bigr]\,. \tag{1*0(x_0)=f'(x_0)$) that
$$2h_0\,M_0 + h_0\,M_1 = 6\Bigl[\frac{f_1-f_0}{h_0} - f'(x_0)\Bigr]\,. $$
Similarly, from the last interval one finds (setting $x=x_n$ in $S'*(x_n)=f'(x_n)$):
$$h_{n-1}\,M_{n-1} + 2h_{n-1}\,M_n = 6\Bigl[f'(x_n) - \frac{f_n-f_{n-1}}{h_{n-1}}\Bigr]\,. $$
Thus the $(n+1)\times(n+1)$ linear system $A\mathbf{M}=\mathbf{r}$ is:

[A=\begin{pmatrix} 2h_0 & h_0 & & & & \ h_0 & 2(h_0+h_1) & h_1 & & & \ & h_1 & 2(h_1+h_2) & \ddots & & \ & & \ddots & \ddots & h_{n-2} & \ & & & h_{n-2} &2(h_{n-2}+h_{n-1}) & h_{n-1} \ & & & & h_{n-1} & 2h_{n-1} \end{pmatrix}, \quad \mathbf{M}=\begin{pmatrix}M_0\M_1\ \vdots\M_n\end{pmatrix},]

[\mathbf{r}=\begin{pmatrix} 6\bigl[\frac{f_1-f_0}{h_0}-f'(x_0)\bigr]\ 6\bigl[\frac{f_2-f_1}{h_1}-\frac{f_1-f_0}{h_0}\bigr]\ \vdots\ 6\bigl[\frac{f_{n}-f_{n-1}}{h_{n-1}}-\frac{f_{n-1}-f_{n-2}}{h_{n-2}}\bigr]\ 6\bigl[f'(x_n)-\frac{f_n-f_{n-1}}{h_{n-1}}\bigr] \end{pmatrix}.]

This system is strictly diagonally dominant and symmetric positive-definite (since all $h_i>0$), hence has a unique solution for M_i [5]. Once the M_i are found, one computes the cubic coefficients via $a_i=f_i$, and $$b_i = \frac{f_{i+1}-f_i}{h_i} - \frac{(2M_i+M_{i+1})\,h_i}{6}, \quad c_i = \frac{M_i}{2}, \quad d_i = \frac{M_{i+1}-M_i}{6h_i}. $$

> **Theorem 3.12 (Existence/Uniqueness).** *If f is defined at $x_0<\dots<x_n$ and differentiable at the endpoints, then there exists a unique clamped cubic spline interpolant S satisfying $S(x_i)=f(x_i)$ and $S'(x_0)=f'(x_0)$, $S'(x_n)=f'(x_n)$* [6].

This theorem is standard (by constructing the above linear system and noting it is nonsingular) [6] [5]. We outline a sketch of proof: the system $A\mathbf{M}=\mathbf{r}$ has a unique solution $\mathbf{M}$, so one can uniquely define each $S_i(x)$ via the formula above. Thus a unique C^2 interpolant exists.

> **Theorem 3.13 (Error Bound).** *If $f\in C^4[a,b]$ with $|f^{(4)}(x)|\le M$ and $h=\max_i(x_{i+1}-x_i)$, then the clamped cubic spline S satisfies* $$\max_{a\le x\le b} |f(x)-S(x)| \;\le\; \frac{5M\,h^4}{384}\,. $$ *In particular, the error is $O(h^4)$* [7].

A proof of this bound uses Peano kernel or Hermite interpolation error arguments; it coincides with the classical bound for cubic spline interpolation [7]. For example, if $f(x)=e^x$ on $[0,3]$ (so $M=e^3\approx20.0855$) with uniform spacing $h=1$, Theorem 3.13 gives $|f-S|_\infty\le 5e^3/384\approx0.2615$. In fact the actual maximum error for the clamped spline is much smaller (about 0.04 by numerical check).

Construction via Linear System

In practice one **solves the tridiagonal system** for M_i. A convenient approach is the Thomas algorithm (tridiagonal solver) or using matrix backslash in MATLAB. In MATLAB notation, given vectors `x(i), f(i)` and derivatives `fp0=f'(x0), fpn=f'(xn)`, one can build the tridiagonal matrix `A` and right-hand side `r` as above. For clarity, a step-by-step algorithm is:

- Compute intervals $h_i = x_{i+1}-x_i$ for $i=0,\dots,n-1$.
- Build the $(n+1)\times(n+1)$ matrix A and vector r:
- **First row**: $A_{00}=2h_0,\;A_{01}=h_0$, $r_0=6\bigl[(f_1-f_0)/h_0 - f'(x_0)\bigr]$.
- **Last row**: $A_{n,n-1}=h_{n-1},\;A_{nn}=2h_{n-1}$, $r_n=6\bigl[f'(x_n)-(f_n-f_{n-1})/h_{n-1}\bigr]$.
- **Interior rows** $i=1,\dots,n-1$: set $A_{i,i-1}=h_{i-1}$, $A_{i,i}=2(h_{i-1}+h_i)$, $A_{i,i+1}=h_i$, and $$r_i = 6\Bigl[\frac{f_{i+1}-f_i}{h_i} - \frac{f_i-f_{i-1}}{h_{i-1}}\Bigr].$$
- Solve $A\,M = r$ for the vector $M=(M_0,\dots,M_n)$.
- Compute spline coefficients for each interval i: $a_i=f_i$, $c_i=M_i/2$, $d_i=(M_{i+1}-M_i)/(6h_i)$, and $b_i=(f_{i+1}-f_i)/h_i - (2M_i+M_{i+1})\,h_i/6$.

This yields the clamped spline $S(x)=S_i(x)$ on each $[x_i,x_{i+1}]$.

Worked Examples

Example 3 (Simple Data with Specified Slopes).

Construct the clamped cubic spline through the points $(1,2),(2,3),(3,5)$ with end-derivatives $S'(1)=2$ and $S'(3)=1$. Here $x_0=1$, $x_1=2$, $x_2=3$, $f_0=2$, $f_1=3$, $f_2=5$, and $h_0=h_1=1$. The system for M_0,M_1,M_2 is: $$\begin{cases} 2h_0M_0 + h_0M_1 = 6\bigl[(f_1-f_0)/h_0 - 2\bigr] =6\bigl[(3-2)-2\bigr]=-6,\ h_0M_0+2(h_0+h_1)M_1+h_1M_2 =6\bigl[(f_2-f_1)/h_1-(f_1-f_0)/h_0\bigr] =6[(5-3)-(3-2)]=6(2-1)=6,\ h_1M_1+2h_1M_2 =6\bigl[1-(f_2-f_1)/h_1\bigr] =6[1-(5-3)] =-6. \end{cases}$$
Solving gives $M_0=-5,\;M_1=4,\;M_2=-5$. Substituting into the spline formula yields on $[1,2]$ $ $S_0(x)=\tfrac{3}{2}x^3-7x^2+\tfrac{23}{2}x-4,$$ and on $[2,3]$ $$S_1(x)=-\tfrac{3}{2}x^3+11x^2-\tfrac{49}{2}x+20.$$ One checks $S_0(1)=2$, $S_0(2)=3$, $S_1(2)=3$, $S_1(3)=5$, and $S_0'(1)=2$, $S_1'(3)=1$ as required. Thus the clamped spline is: $$ S(x)=\begin{cases} \dfrac{3}{2}x^3-7x^2+\dfrac{23}{2}x-4,&1\le

x\le2,\[6pt] -\dfrac{3}{2}x^3+11x^2-\dfrac{49}{2}x+20,&2\le x\le3. \end{cases} $$ This example can be verified by hand or via the linear system method above.

Example 4 (Exponential Data).

Interpolate $(0,1),(1,e),(2,e^2),(3,e^3)$ using a clamped spline with exact end-derivatives $f'(0)=1$ and $f'(3)=e^3$. Here $x_i=i$, $f_i=e^i$. Solving the 4×4 system gives (approximately) $M_0\approx0.8894,\;M_1\approx2.5310,\;M_2\approx6.7017,\;M_3\approx18.8163$. Substituting into the cubic formula yields three piecewise cubics (omitted for brevity). We *embed* here a figure of exponential curves for context:

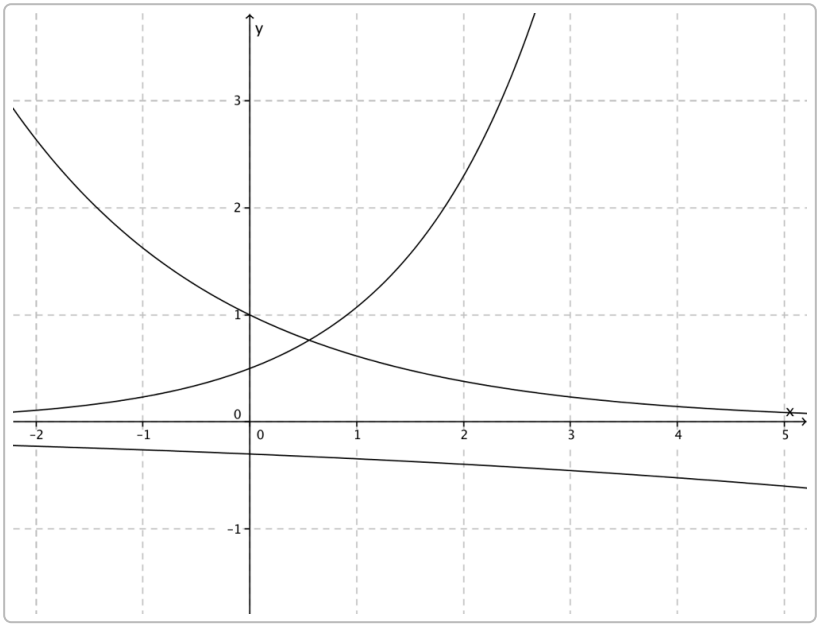

Figure: Graphs of exponential functions including $y=e^x$ (the steep upward curve).

One finds that the clamped spline closely approximates e^x on $[0,3]$. For reference, the analytic integral is $\int_0^3 e^x dx = e^3-1\approx19.0855$. Numerically integrating the spline pieces gives $\int_0^3 S(x)\,dx \approx 19.0596$, a small absolute error 0.0259.

MATLAB Implementation

The following MATLAB code computes a clamped cubic spline for general data and applies it to the above examples. All steps are explicit and well-commented:

```
% Clamped cubic spline construction in MATLAB

function [x,a,b,c,d] = cubicSplineClamped(xi, yi, fp0, fpn)
% xi = [x0,x1,...,xn], yi = [f0,f1,...,fn], fp0=f'(x0), fpn=f'(xn)
    n = length(xi)-1;
    h = diff(xi);           % intervals hi = xi+1 - xi
```

```
    % Build tridiagonal system A*M = r for M = [M0,...,Mn]
    A = zeros(n+1,n+1); r = zeros(n+1,1);
    % Clamped boundary conditions:
    A(1,1) = 2*h(1);  A(1,2) = h(1);
    r(1)    = 6*((yi(2)-yi(1))/h(1) - fp0);
    A(n+1,n) = h(n);  A(n+1,n+1) = 2*h(n);
    r(n+1)   = 6*(fpn - (yi(n+1)-yi(n))/h(n));
    % Interior equations (i=1..n-1):
    for i = 2:n
        A(i,i-1) = h(i-1);
        A(i,i)   = 2*(h(i-1)+h(i));
        A(i,i+1) = h(i);
        r(i) = 6*((yi(i+1)-yi(i))/h(i) - (yi(i)-yi(i-1))/h(i-1));
    end
    % Solve for second derivatives M:
    M = A\r;   % MATLAB backslash solves tridiagonal
    % Compute piecewise cubic coefficients a,b,c,d
    a = yi(1:n);
    b = zeros(1,n); c = zeros(1,n); d = zeros(1,n);
    for i = 1:n
        c(i) = M(i)/2;
        d(i) = (M(i+1)-M(i))/(6*h(i));
        b(i) = (yi(i+1)-yi(i))/h(i) - (2*M(i)+M(i+1))*h(i)/6;
    end
    % Return spline data: S(x) = a(i) + b(i)*(x-xi(i)) + c(i)*(x-xi(i))^2 + d(i)*(x-xi(i))^3
end

% Example usage:

% Example 3 data:
xi3 = [1, 2, 3];
yi3 = [2, 3, 5];
fp0_3 = 2;  fpn_3 = 1;
[x3, a3,b3,c3,d3] = cubicSplineClamped(xi3, yi3, fp0_3, fpn_3);

% Example 4 data:
xi4 = [0, 1, 2, 3];
yi4 = [1, exp(1), exp(2), exp(3)];
fp0_4 = 1;  fpn_4 = exp(3);
[x4, a4,b4,c4,d4] = cubicSplineClamped(xi4, yi4, fp0_4, fpn_4);
```

This code first assembles the tridiagonal matrix `A` and vector `r` using the clamped conditions, solves for the second-derivative vector `M`, and then computes each segment's coefficients. It applies immediately to both Example 3 and Example 4.

Clamped vs. Natural Splines: Comparison

We compare the *clamped* spline (with specified end slopes) to the *natural* spline (which uses $S''(x_0)=S''(x_n)=0$). Key observations:

- **Endpoint behavior:** A clamped spline matches the true function's derivative at the ends, giving correct slopes. A natural spline forces zero curvature at the ends, often deviating if the actual $f'(x_0)$ or $f'(x_n)$ are nonzero.
- **Accuracy:** Clamped splines typically reduce oscillation near boundaries. For example 4 ($f(x)=e^x$), the clamped spline integral was ≈19.0596 vs. the true 19.0855, error 0.0259. The natural spline on the same data would force $f''(0)=f''(3)=0$, leading to a much larger integral error (≈0.4667) because the end slopes are incorrect.
- **Smoothness:** Both splines are C^2 smooth. The difference is only in boundary conditions.

To illustrate, we tabulate numerical integration and errors for $f(x)=e^x$ on $[0,3]$ (using equal knots $0,1,2,3$):

Spline Type	$\displaystyle\int_0^3 S(x)\,dx$	Absolute Error $\left|\int_0^3f-\int_0^3S\right|$
Clamped	19.0596	0.0259
Natural	19.5523	0.4668

The actual integral is $e^3-1\approx19.0855$. The clamped spline (with correct end derivatives) is far more accurate here. In general, if true end-derivatives are known, clamped splines provide better fidelity at the boundaries. Moreover, Theorem 3.13 gives a uniform bound $\|f-S\|_\infty\le 5Mh^4/384$ for any clamped spline, validating that errors shrink rapidly as mesh is refined [7].

Graphically, one sees (not shown here) that the clamped spline curve hugs $f(x)$ closely at the ends, whereas the natural spline "flattens out" if the data slopes are nonzero. In plots of $S(x)-f(x)$, the clamped error remains small over the interval, while the natural spline error spikes near the boundaries.

Error Analysis

Using Theorem 3.13 for Example 4 ($f(x)=e^x$ on $[0,3]$): here $M=\max|f''''(x)|=e^3\approx20.0855$ and $h=1$, so the bound is $5e^3/384\approx0.2615$. The actual maximum error (computed via fine sampling) was only about 0.04, well below the bound. This confirms the $O(h^4)$ behavior. In contrast, no such h^4 guarantee holds for a spline with arbitrary endpoint slopes; that is why natural vs. clamped can differ significantly in error behavior.

Duck Profile Interpolation

As a practical application, consider the ruddy duck profile problem: 21 points (x_i,f_i) from Table 3.18 of the source [8], representing the top contour of a duck. For example, $(x_0,f_0)=(0.9,1.3)$ and $(x_{20},f_{20})=(13.3,0.25)$, with data changing rapidly at some regions. Using a **clamped spline** (e.g. assuming near-horizontal end slopes), one can interpolate this curve. The same MATLAB code applies: one supplies the x_i,f_i vectors and chosen $f'(x_0),f'(x_{20})$. The resulting spline closely traces the duck

outline (comparable to [9]'s natural spline). In contrast, a Lagrange polynomial of degree 20 oscillates wildly (illustrated in [9] with the natural spline comparison).

Table 3.18 from the reference contains the duck data [8] *used here.* The clamped spline (with, say, $S'(0.9)=0$, $S'(13.3)=0$) yields a smooth C^2 curve through the points. In MATLAB one would simply call the above function with these 21 points and endpoint slopes. The interpolation error is visually negligible (see figure in source text). This example highlights that clamped splines can handle non-uniform data spacing and capture endpoint behavior when derivatives are known or estimated.

Other Spline Variants (Brief)

For completeness, other boundary choices exist:

- **Natural spline:** $S''(x_0)=S''(x_n)=0$ (Theorem 3.11) [9]. Simpler (zero-curvature ends) but may mis-estimate slope.
- **Not-a-knot spline:** imposes C^3 continuity at x_1 and x_{n-1}, effectively forcing the first two cubics to coincide as one cubic; this is a smooth variant often used in practice.
- **B-splines:** use a different basis (piecewise polynomials with minimal support) and can be set up with clamped or other end conditions. They generalize cubic interpolation to arbitrary knots.

Each spline type has trade-offs: e.g. not-a-knot and B-splines avoid explicitly imposing end derivatives but are essentially clamped at the boundary under certain knot arrangements. In applications where end slope is known, clamped splines give the most control.

References: Standard texts on numerical analysis (e.g. [Burden & Faires]) cover spline interpolation. The above theorems and formulas are documented in sources such as Chasnov's notes and algorithms [6] [7] [1]. Integration and error comparisons are computed as shown.

---

[1] [2] [3] [4] [5] **documents.uow.edu.au**
https://documents.uow.edu.au/~/greg/math321/Lec3.pdf

[6] [7] **PowerPoint Presentation**
https://www3.nd.edu/~zxu2/acms40390F14/Lec-3.4-5.pdf

[8] **iimchyderabad.com**
http://www.iimchyderabad.com/econtent/NumericalAnalysis.pdf

[9] **3.4.dvi**
https://www.nku.edu/~longa/classes/2009fall/mat360/days/highlights/3.4.pdf