

Piecewise Polynomial Interpolation

Piecewise linear interpolation

Suppose we have data point (x_k, y_k) , $k = 0, 1, \dots, N$. A piecewise linear polynomial that interpolates these points is given by

$$p(x) = p_k(x), \quad x \in [x_k, x_{k+1}]$$

where the polynomials $p_k(x)$ can be written as

$$p_k(x) = a_1(x - x_k) + a_0, \quad k = 1, 2, \dots, N - 1$$

At this point, we could easily compute the coefficients $\mathbf{a} = [a_1, a_0]$, but let's go through a more general procedure that we can use for higher order piecewise polynomials.

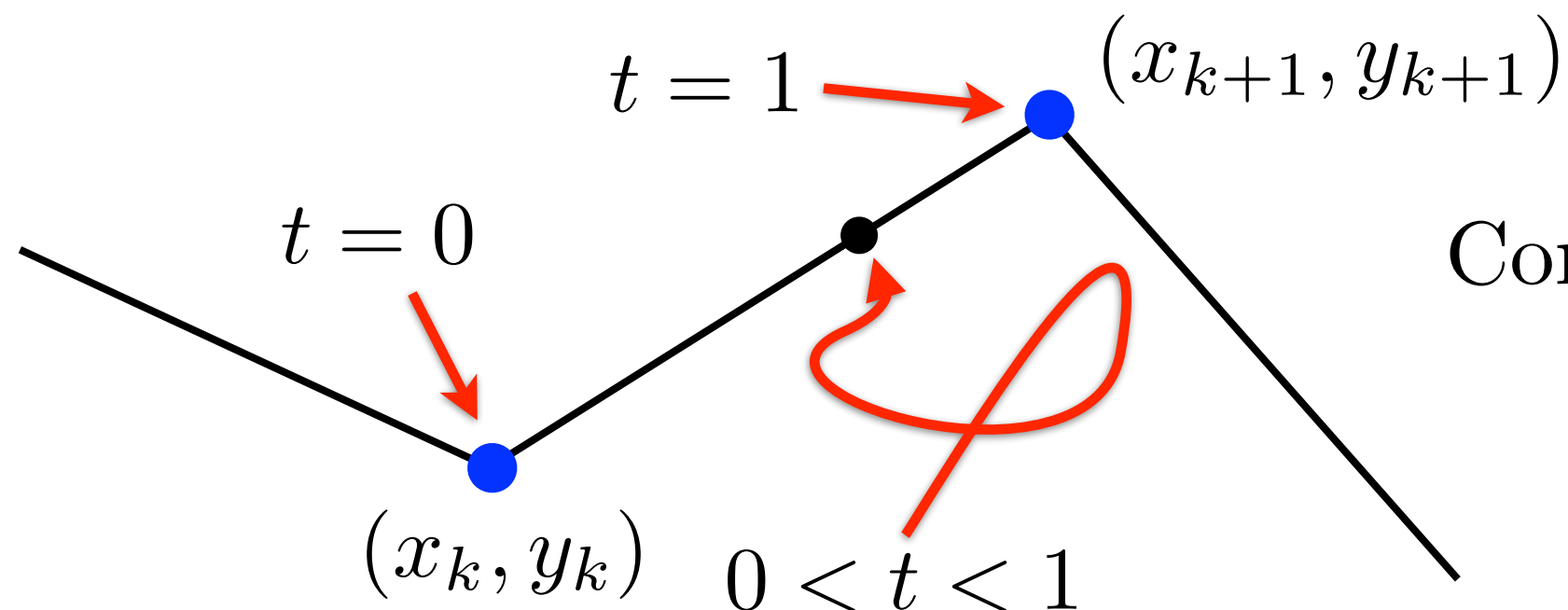
Piecewise linear interpolation

We can write this as a function of a local variable t which measures the distance along the interval.

$$t = \frac{x - x_k}{x_{k+1} - x_k} \quad x = h_k t + x_k$$

where $h_k = x_{k+1} - x_k$. Then, for $0 \leq t \leq 1$, we have

$$p_k(x) = \tilde{p}_k(t) = \tilde{a}_1 t + \tilde{a}_0$$



Constraints to impose

$$\tilde{p}_k(0) = y_k$$

$$\tilde{p}_k(1) = y_{k+1}$$

Piecewise linear interpolation

This leads to a linear system

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \tilde{a}_1 \\ \tilde{a}_0 \end{bmatrix} = \begin{bmatrix} y_k \\ y_{k+1} \end{bmatrix}$$

which we write as $A\tilde{\mathbf{a}} = \mathbf{p}$. To solve for the coefficients, we use the matrix *co-factor* C to get A^{-1}

$$A^{-1} = \frac{C^T}{\det(A)} = \frac{1}{\det(A)} \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix}$$

We have $\det(A) = -1$, and so

$$\tilde{\mathbf{a}} = -C^T \mathbf{p}$$

Piecewise linear interpolation

To see what $\tilde{p}_k(t)$ looks like, we let $\mathbf{q}(t) = [t, 1]$ and write

$$\begin{aligned}\tilde{p}_k(t) &= \tilde{\mathbf{a}}^T \mathbf{q}(t) \\ &= -\mathbf{p}^T C \mathbf{q}(t) \\ &= - \begin{bmatrix} y_k & y_{k+1} \end{bmatrix}^T \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix}\end{aligned}$$

The advantage of this form is that we can clearly see how the polynomial $\tilde{p}_k(t)$ depends on our data y_k and y_{k+1} :

$$\tilde{p}_k(t) = (1 - t)y_k + ty_{k+1}$$

Piecewise linear interpolation

To build a spline in Matlab, we need to have the coefficients a_0 and a_1 of our original polynomial.

$$p_k(x) = a_1(x - x_k) + a_0$$

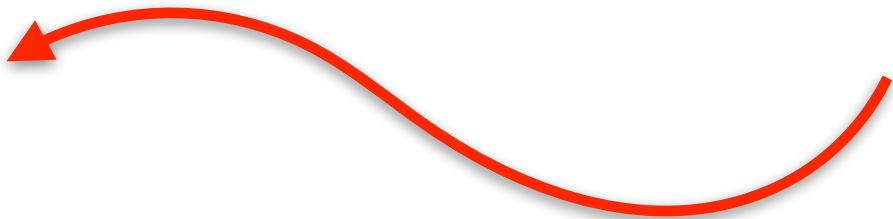
We can recover these coefficients by substituting $t = (x - x_k)/h_k$ into $\tilde{p}_k(t) = \tilde{a}_1 t + \tilde{a}_0$ to get

$$p_k(x) = \tilde{a}_1 \left(\frac{x - x_k}{h_k} \right) + \tilde{a}_0 = a_1(x - x_k) + a_0$$

where $a_1 = \tilde{a}_1/h_k$ and $\tilde{a}_0 = a_0$.

Piecewise linear interpolation

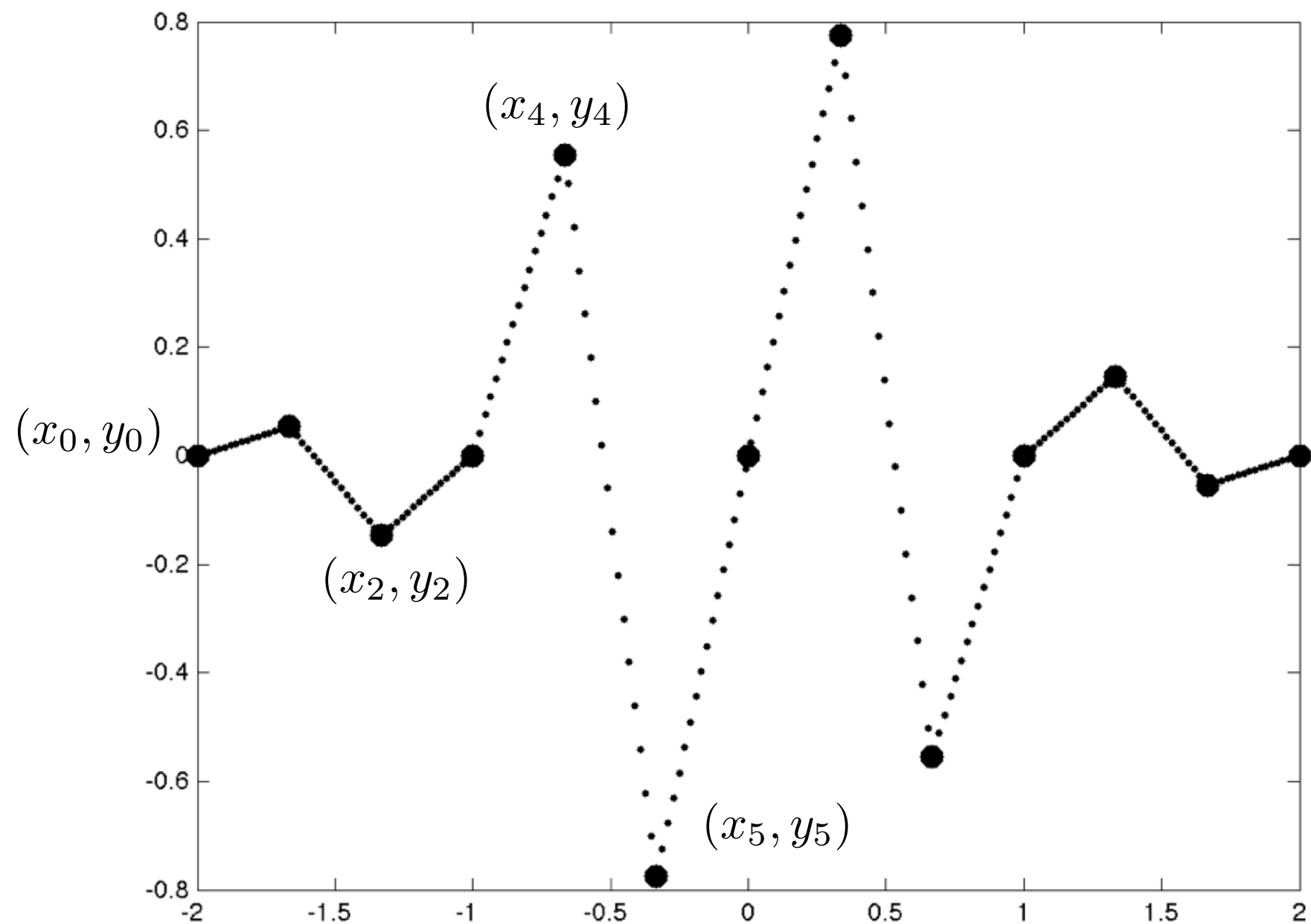
We can also write $\mathbf{a} = [a_1, a_0]$ using

$$\begin{aligned}\mathbf{a} &= -\mathbf{p}^T \mathbf{C} \mathbf{S} = - \begin{bmatrix} y_k & y_{k+1} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1/h_k & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{y_{k+1} - y_k}{h_k} \\ y_k \end{bmatrix}\end{aligned}$$


These are exactly the coefficients that Matlab needs to build a spline using 'mkpp'.

```
function pp_linear = build_linear_spline(xdata,ydata)
N = length(xdata)-1;
h = diff(xdata);    %(x1-x0, x2-x1, x3-x2,...)
C = [1 -1; -1 0];
for k = 1:N,
    p = [ydata(k); ydata(k+1)];
    S = diag([1./h(k) 1]);
    coeffs(k,:) = -p'*C*S;
end
pp_linear = mkpp(xdata,coeffs);
end
```

Piecewise linear interpolation



Piecewise linear interpolation

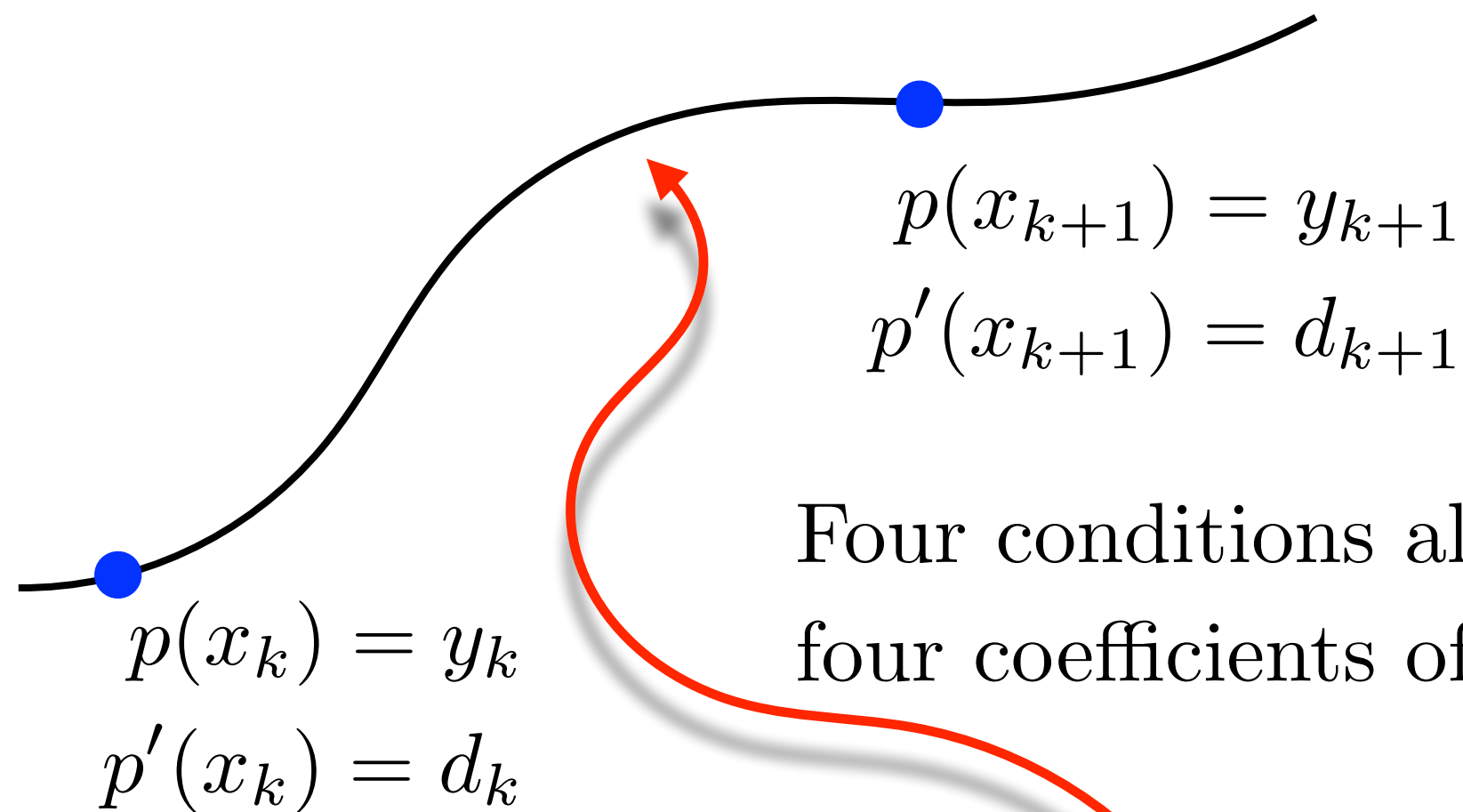
Piecewise linear interpolation has these properties

- *Continuous* at nodes (x_k, y_k) .
- Derivatives are not continuous (the function is not smooth!)

Can we come design a method that matches derivatives at the nodes as well?

Hermite interpolation

We can impose *smoothness* at the nodes to get a nicer looking curve. Suppose we match the function values *and* derivatives.



This approach assumes that we have derivatives d_k at each of the nodes x_k .

Four conditions allow us to find the four coefficients of a cubic polynomial

$$p_k(x) = a_3(x - x_k)^3 + a_2(x - x_k)^2 + a_1(x - x_k) + a_0$$

Hermite interpolation

As before, define

$$t = \frac{x - x_k}{x_{k+1} - x_k}, \quad \rightarrow \quad x = h_k t + x_k$$

where $h_k = x_{k+1} - x_k$. Then, for $0 \leq t \leq 1$, we have

$$p_k(x) \equiv \tilde{p}_k(t) = \tilde{a}_3 t^3 + \tilde{a}_2 t^2 + \tilde{a}_1 t + \tilde{a}_0$$

and our four conditions become

$$\begin{aligned} \tilde{p}_k(0) &= y_k & \tilde{p}_k(1) &= y_{k+1} \\ \tilde{p}'_k(0) &= d_k h_k & \tilde{p}'_k(1) &= d_{k+1} h_k \end{aligned}$$

Hermite interpolation

This leads to the following linear system for the coefficients \tilde{a}_n :

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} \tilde{a}_3 \\ \tilde{a}_2 \\ \tilde{a}_1 \\ \tilde{a}_0 \end{bmatrix} = \begin{bmatrix} y_k \\ y_{k+1} \\ d_k h_k \\ d_{k+1} h_k \end{bmatrix}$$

We write the above system as $A\tilde{\mathbf{a}} = \mathbf{p}$. To invert A , we use the *co-factor* matrix C to get

$$A^{-1} = \frac{C^T}{\det(A)}$$

See Strang's *Introduction to Linear Algebra* for more information on the co-factor formula for the matrix inverse.

Hermite interpolation

We can then write $\tilde{p}_k(t)$ as

$$\tilde{p}_k(t) = \tilde{\mathbf{a}}^T \mathbf{q}(t) = -\mathbf{p}^T C \mathbf{q}(t)$$

where $\mathbf{q}(t) = [t^3, t^2, t, 1]^T$ and we have used the fact that $\det(A) = -1$. Computing C , we get

$$\tilde{p}_k(t) = - \begin{bmatrix} y_k & y_{k+1} & d_k h_k & d_{k+1} h_k \end{bmatrix}^T \begin{bmatrix} -2 & 3 & 0 & -1 \\ 2 & -3 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ -1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

which leads to a formula that clearly shows how the polynomial depends on our data y_k , y_{k+1} , d_k and d_{k+1} :

$$\begin{aligned} \tilde{p}_k(t) &= [1 - t^2(3 - 2t)] y_k + [t^2(3 - 2t)] y_{k+1} \\ &+ [t(t - 1)^2] h_k d_k + [t^2(t - 1)] h_k d_{k+1} \end{aligned}$$

Piecewise Cubic Hermite Interpolating Polynomial

$$\begin{aligned}\tilde{p}_k(t) &= [1 - t^2(3 - 2t)] y_k + [t^2(3 - 2t)] y_{k+1} \\ &+ [t(t - 1)^2] h_k d_k + [t^2(t - 1)] h_k d_{k+1}\end{aligned}$$

If we substitute $t = (x - x_k)/h_k$, we can recover our original polynomial in x :

$$p_k(x) = a_3(x - x_k)^3 + a_2(x - x_k)^2 + a_1(x - x_k) + a_0$$

where $a_n = \tilde{a}_n/h_k^n$. We can write these coefficients as

$$\mathbf{a} = -\mathbf{p}^T C S$$

These are exactly the coefficients that Matlab needs to build a spline using 'mkpp'.

where $S = \text{diag}(h_k^{-3}, h_k^{-2}, h_k^{-1}, 1)$, $\mathbf{p} = [y_k, y_{k+1}, d_k h_k, d_{k+1} h_k]$, and C is the co-factor matrix of our original coefficient matrix A .

The derivatives?

Where do we get the derivatives d_k from?

- If we know $f(x)$, we can set $d_k = f'(x_k)$.
- We can approximate derivatives using data (x_k, y_k) .

$$d_k = G(\delta_{k-1}, \delta_k), \quad \delta_k = \frac{y_{k+1} - y_k}{h_k}$$

where, for example, we could take $G(a, b) = \frac{a+b}{2}$, $G(a, b) = a$, $G(a, b) = b$ or some other reasonable function. Matlab's PCHIP chooses derivatives to avoid introducing new extrema in the spline.

- We can impose additional smoothness conditions and solve implicitly for the derivatives.

“PCHIP” derivatives

Explicit formula for d_k , $k = 0, 1, 2, \dots, N$ that avoids non-physical extrema.

Set $h_k = x_{k+1} - x_k$, $k = 0, 1, \dots, N - 1$.

At each interior point $k = 1, 2, 3, \dots, N - 1$, compute a slope d_k as follows.

- If δ_{k-1} and δ_k have different signs, set $d_k = 0$.
- If δ_{k-1} and δ_k have the same sign, then

$$\frac{w_1 + w_2}{d_k} = \frac{w_1}{\delta_{k-1}} + \frac{w_2}{\delta_k}$$

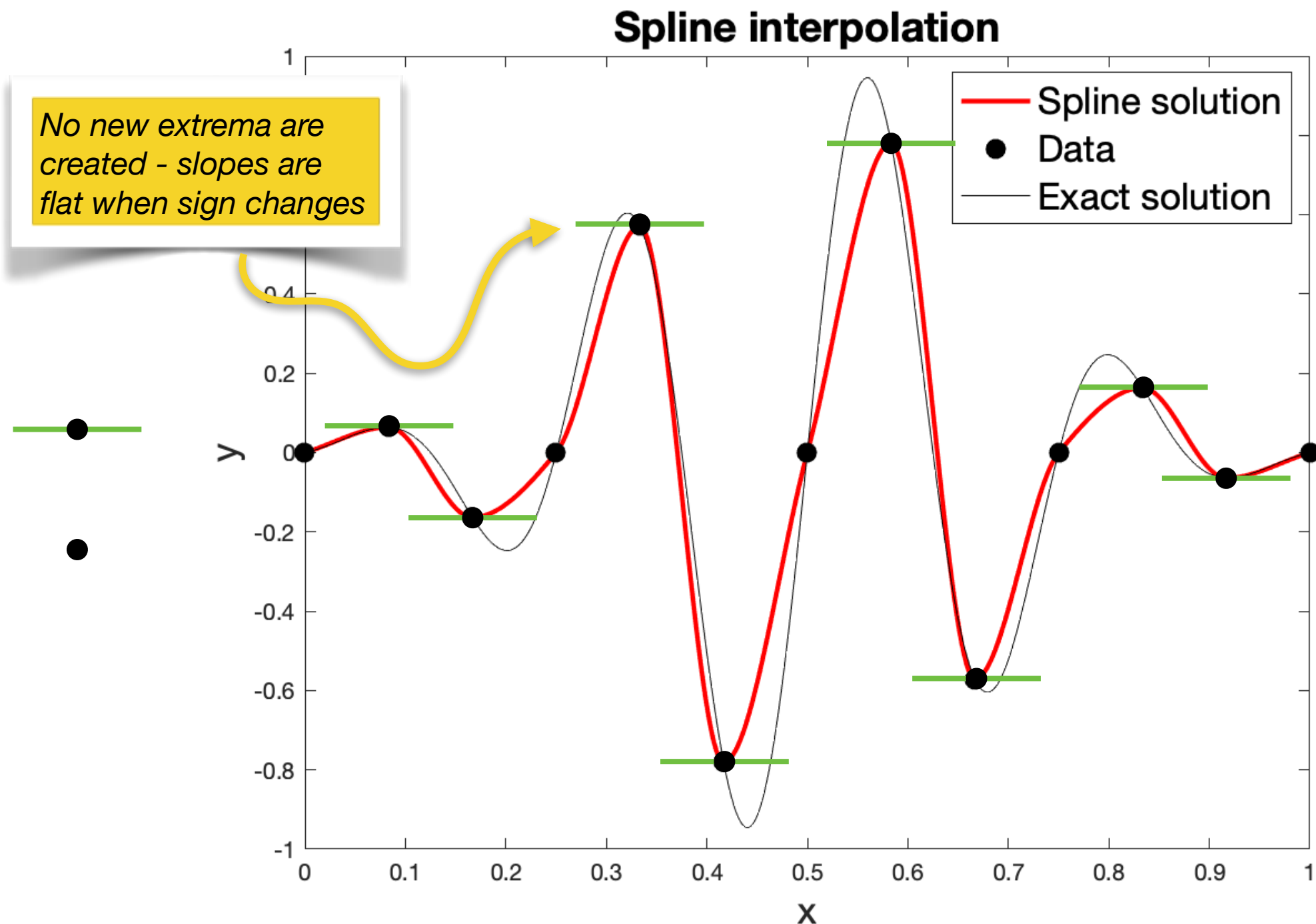
where

$$w_1 = 2h_k + h_{k-1} \quad \text{and} \quad w_2 = h_k + 2h_{k-1}$$

Use the above to solve for d_k , $k = 1, 2, \dots, N - 1$. Set the derivatives at the end points to $d_0 = \delta_0$ and $d_N = \delta_{N-1}$.

The derivatives above are those that are used in the Matlab PCHIP routine or the SciPy routine [scipy.interpolate.PchipInterpolator](https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.PchipInterpolator.html).

Using PCHIP derivatives



$$f(x) = e^{-15(x-0.5)^2} \sin(8\pi x)$$

Classic spline derivatives

In another approach, additional constraints are imposed.

Given data points $(x_0, y_0), \dots, (x_N, y_N)$, impose continuity in the second derivative at the interior nodes,

$$p_k''(x_{k+1}) = p_{k+1}''(x_{k+1}), \quad k = 1, 2, \dots, N-1.$$

This leads to a tridiagonal system for the derivatives at internal nodes ("knots") $d_1, d_2, d_3, \dots, d_{N-1}$.

To get equations for the end point derivatives, we will need to impose additional conditions at the endpoints.

Piecewise linear interpolation

The entries in the tridiagonal system can be found by considering the polynomials as functions of t :

$$p''_{k-1}(x) = \frac{1}{h_{k-1}^2} \tilde{p}''_{k-1}(t), \quad x = h_{k-1}t + x_{k-1}$$

Transforming the equations equating second derivatives at knots, we obtain

$$\frac{1}{h_{k-1}^2} \tilde{p}''_{k-1}(1) = \frac{1}{h_k^2} \tilde{p}''_k(0), \quad k = 1, 2, 3, \dots, N-1$$

Using $\tilde{p}''_k(t) = -\mathbf{p}^T C \mathbf{q}''(t)$, we get

$$\frac{6}{h_{k-1}^2} y_{k-1} - \frac{6}{h_{k-1}^2} y_k + \frac{2}{h_{k-1}} d_{k-1} + \frac{4}{h_{k-1}} d_k = \frac{6}{h_k^2} y_k - \frac{6}{h_k^2} y_{k+1} + \frac{2}{h_k} d_k + \frac{4}{h_k} d_{k+1}$$

or

$$\frac{1}{h_{k-1}} d_{k-1} + 2 \left(\frac{1}{h_{k-1}} + \frac{1}{h_k} \right) d_k + \frac{1}{h_k} d_{k+1} = \frac{3}{h_{k-1}^2} (y_k - y_{k-1}) + \frac{3}{h_k^2} (y_{k+1} - y_k)$$

for $k = 1, 2, 3, \dots, N-1$. This leads to a tridiagonal system for the unknown derivatives d_k .

Piecewise linear interpolation

We can combine all equations into an $(N - 1) \times (N + 1)$ "augmented" tridiagonal system $A'\mathbf{d} = \mathbf{b}$, or

$$\left[\begin{array}{c|cccccc} u_0 & 2(u_0 + u_1) & u_1 & & & \\ & u_1 & 2(u_1 + u_2) & u_2 & & \\ & & u_2 & 2(u_2 + u_3) & u_3 & \\ & & & \ddots & \ddots & \ddots \\ & & & & u_{N-2} & 2(u_{N-2} + u_{N-1}) & u_{N-1} \end{array} \right] \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{N-1} \\ d_N \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_{N-1} \end{bmatrix}$$

where the matrix A'

$$A' = [\mathbf{u}_0 : A : \mathbf{u}_N]$$

and

$$u_i = \frac{1}{h_i}, \quad k = 0, 1, 2, \dots, N - 1$$

$$b_i = 3u_{i-1}^2(y_i - y_{i-1}) + 3u_i^2(y_{i+1} - y_i), \quad k = 1, 2, 3, \dots, N - 1.$$

We can write this as a square tridiagonal system

$$A\mathbf{d} = \mathbf{b} - d_0\mathbf{u}_0 - d_N\mathbf{u}_N$$

where \mathbf{u}_0 and \mathbf{u}_N are the first and last columns of A' above, and $\mathbf{d} = (d_1, d_2, \dots, d_{N-1})$

Piecewise linear interpolation

We solve for the unknown derivatives $d_0, d_1, d_2, \dots, d_N$ in three steps.

1. Solve $\mathbf{d}_H = A^{-1}\mathbf{b}$.

2. Solve

$$A\mathbf{d}_0 = \mathbf{u}_0$$

$$A\mathbf{d}_N = \mathbf{u}_N$$

for vectors \mathbf{d}_0 and \mathbf{d}_N .

3. Solve

$$\mathbf{F}(d_0, d_N) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} d_0 \\ d_N \end{bmatrix} + \begin{bmatrix} F_0 \\ F_N \end{bmatrix} = 0$$

for scalars d_0, d_N so that so that "endpoint" conditions (to be determined) are satisfied. Coefficients for the above system depend on choice of endpoint conditions.

All unknown derivatives are then given by

$$\mathbf{d} = \{ d_0, \quad \mathbf{d}_H - d_0\mathbf{d}_0 - d_N\mathbf{d}_N, \quad d_N \}$$

End conditions?

There are several additional conditions we can impose to solve for the endpoint derivatives.

- *Clamp* derivative values d_0 and d_N to fixed values,
- Fit a cubic through first (last) four points and set d_0 (d_N) to the derivative of this polynomial at x_0 (x_N).
- Impose continuity of the third derivative at x_1 (x_{N-1}). This is the "not-a-knot" condition, since we now have a single cubic polynomial over $[x_0, x_2]$ and $[x_{N-2}, x_N]$, effectively eliminating the second (and second to last) node (or "knot").
- For a periodic spline, match first and second derivatives between x_0 and x_N .
- Impose the 'natural' endpoint conditions, i.e. second derivative equal to 0.

Each of these can be represented as a 2×2 system of the form

$$\mathbf{F}(d_0, d_N) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} d_0 \\ d_N \end{bmatrix} + \begin{bmatrix} F_0 \\ F_N \end{bmatrix} = 0$$

After determining coefficients, we can then solve for d_0 and d_N .

Piecewise linear interpolation

Example : The function $F(d_0, d_N)$ for the "natural" endpoint condition is given by :

$$\begin{aligned} F_1 &\equiv p_0''(x_0) \\ F_2 &\equiv p_{N-1}''(x_N) \end{aligned}$$

To compute the derivatives, use the coefficients that describe the polynomials in each of the two segments.

$$\begin{aligned} F_1 &\equiv 2a_2 \\ F_2 &\equiv 6b_3(x_N - x_{N-1}) + 2b_2 \end{aligned}$$

See the [coefficient formula](#)

The dependence on d_0 and d_N is "hidden" in the coefficients.

Note: It is not necessary to get an explicit formula for \mathbf{F} in terms of d_0 and d_N . You only need a way to compute the coefficients given a set of derivatives $d_0, d_1, d_2, \dots, d_N$.

Piecewise linear interpolation

To solve $\mathbf{F}(\mathbf{d}) = 0$, we will use Newton's Method. In order to do this, we need $J \equiv F'(\mathbf{d})$, the Jacobian of \mathbf{F} :

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial d_0} & \frac{\partial F_1}{\partial d_N} \\ \frac{\partial F_2}{\partial d_0} & \frac{\partial F_2}{\partial d_N} \end{bmatrix}$$

We can compute the Jacobian J using a *finite difference* idea from Calculus I. For example, column 1 of J can be computed as:

$$\frac{\partial \mathbf{F}}{\partial d_0} = \frac{\mathbf{F}(\mathbf{d} + [h; 0]) - \mathbf{F}(\mathbf{d})}{h}$$

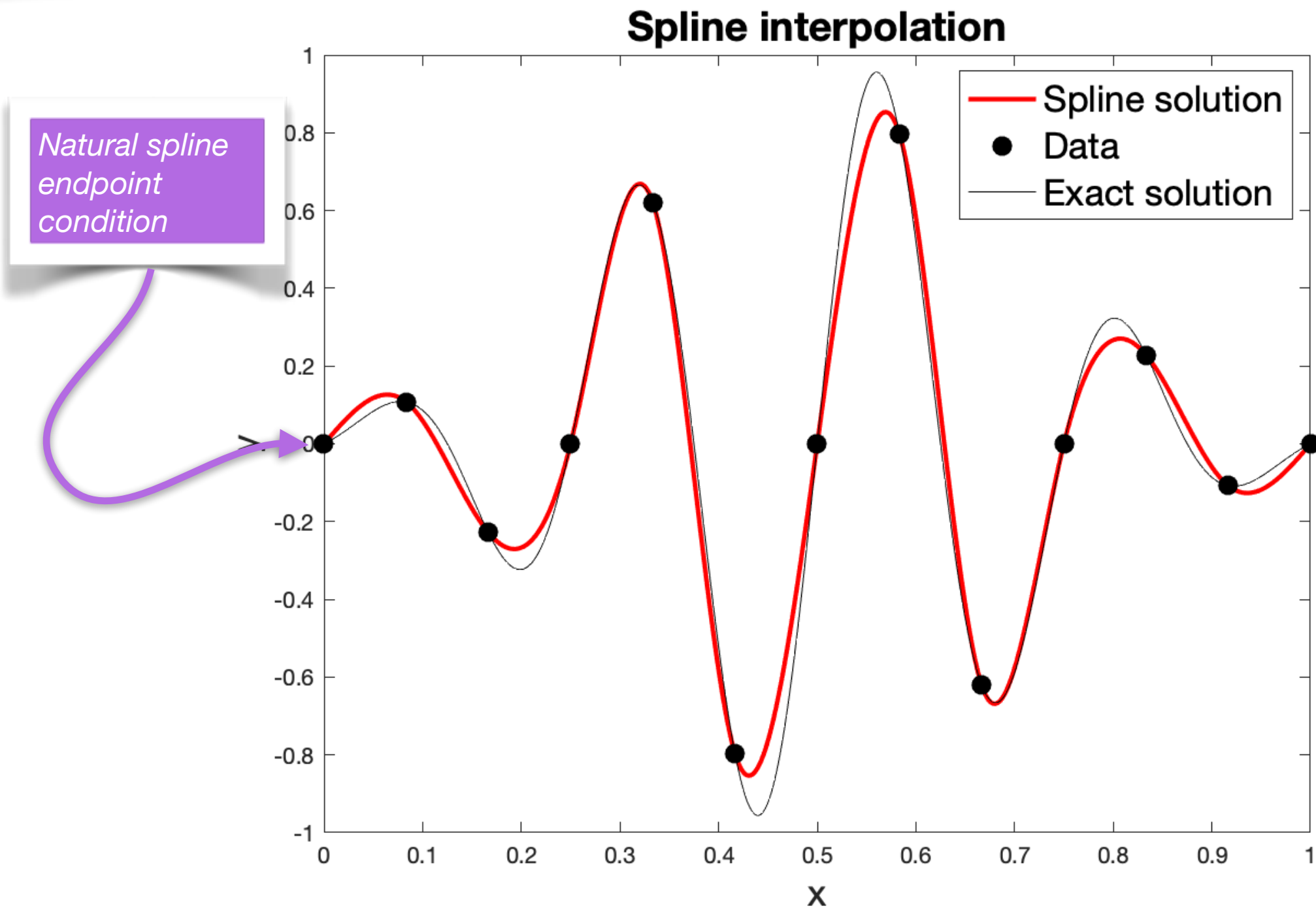
where we can take $\mathbf{d} = [0; 0]$ and $h = 1$. The second column of J can be computed in an analogous way. The Newton step is then given by

$$d^{k+1} = d^k - J^{-1} \mathbf{F}(d_k)$$

Homework question: We expect this to converge in one step. Why?

Once we have the correct values for \mathbf{d} , we use the coefficient formula to compute the final coefficients for the spline.

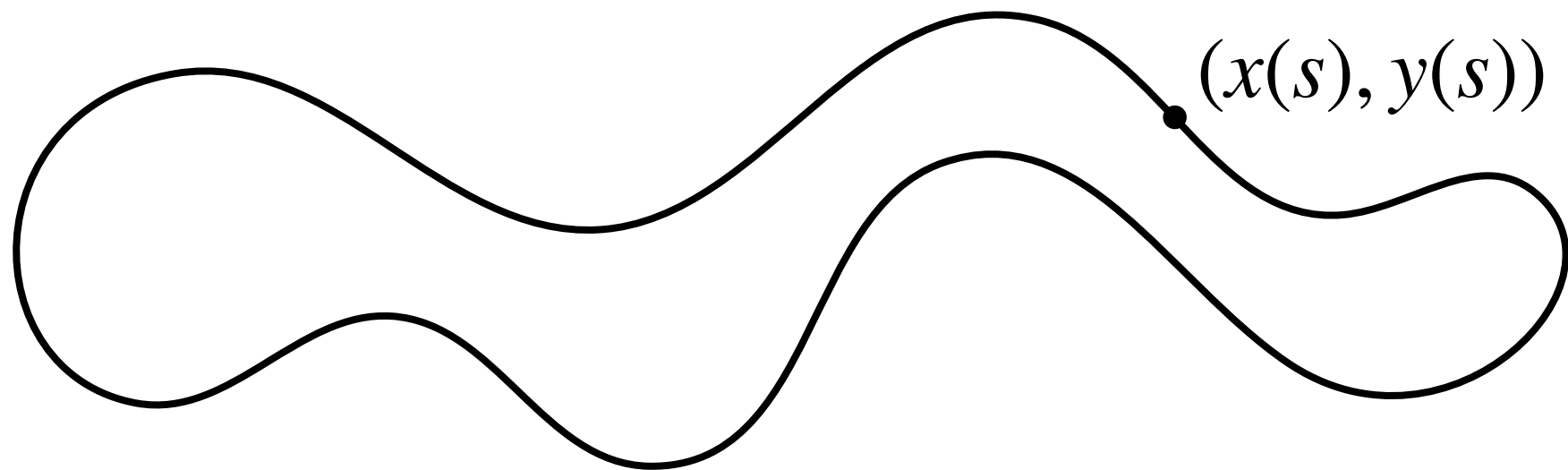
Splines derivatives



$$f(x) = e^{-15(x-0.5)^2} \sin(8\pi x)$$

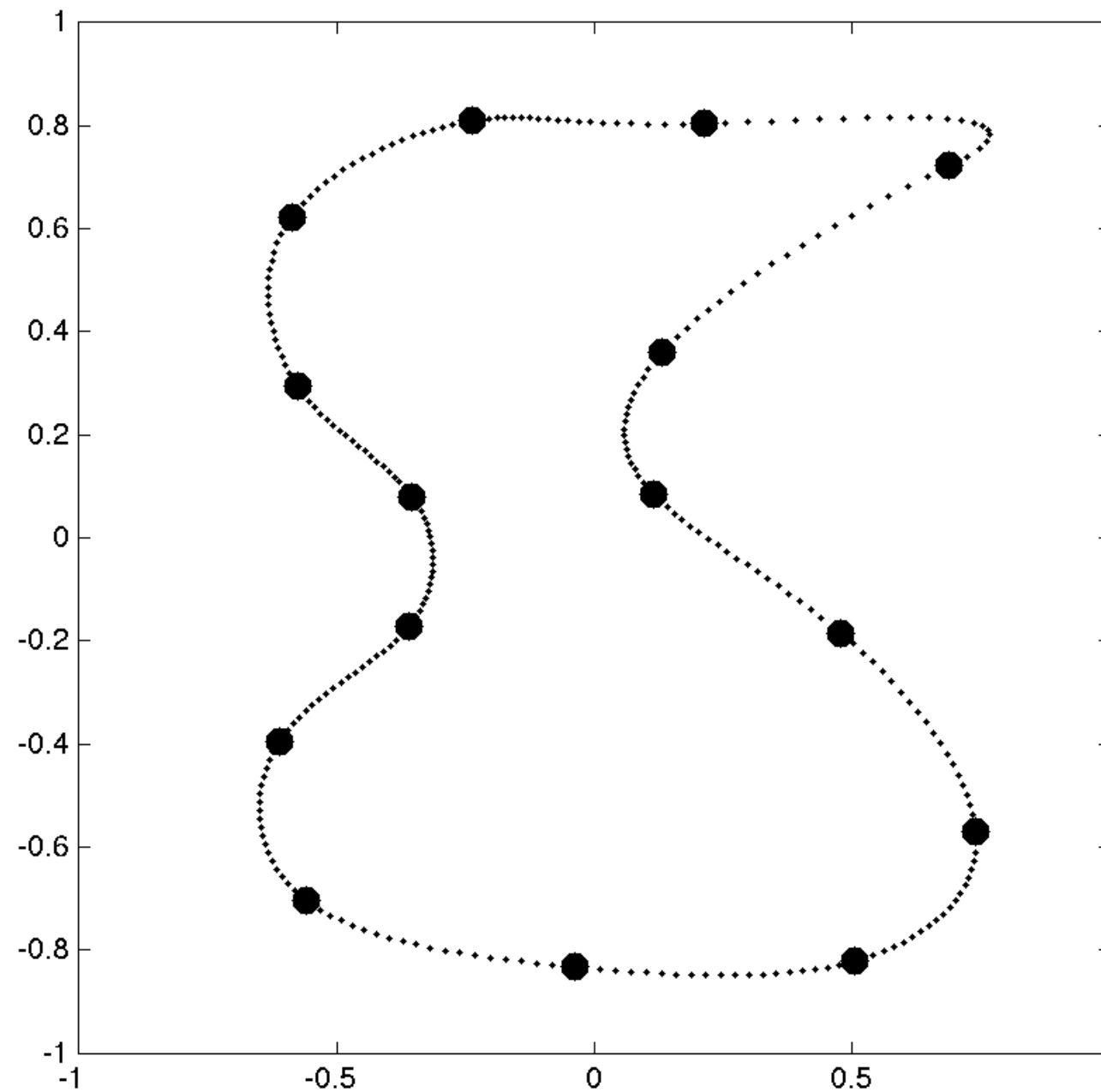
Interpolating general data

For data (x_k, y_k) , where y_k is not a function of x_k , we have to introduce an artificial independent variable s_k where s_k can be taken as any monotonically increasing sequence.



For example, choose $s_k = k$ and compute splines for data (k, x_k) and (k, y_k) separately. For closed curves, use periodic boundary conditions. Another choice of independent variable is *arclength*, but this is often difficult to get.

Periodic spline



Periodic endpoint conditions

Useful routines for splines

Several routines greatly simplify creating your own splines.

In Matlab

`mkpp, ppval, histcounts`

In Python

[scipy.interpolate.CubicHermiteSpline](#)

Splines are fun!

