The Interpolation Problem in 1D

John Burkardt
Department of Scientific Computing
Florida State University
http://people.sc.fsu.edu/~jburkardt/presentations/...
...ornl_interp_1d_2012.pdf

20 August 2012

The Interpolation Problem in 1D

- **Introduction**
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

The Interpolation Problem in 1D

I have been working for some time in the area of sparse grid quadrature rules for integrands in high dimensions.

Recently, it has become important to try to extend this work to the related problem of sparse grid interpolation.

Now sparse grids quadrature rules are linear combinations of product quadrature rules, which are products of 1D quadrature rules, which are often formed by constructing an interpolant to data and integrating it.

So in some sense, I simply have to walk backwards to the 1D interpolation problem, and then "carry it forward" into higher dimensions without taking the quadrature step.

The Interpolation Problem in 1D

I am actually interested in the interpolation problem in high dimension; however, both high dimensionality and interpolation, separately, have tricky features, which, when combined, can create disastrous problems.

Therefore, in this talk, I want to warm up to the high dimensional problem by going over some natural interpolation approaches in the lowest possible dimension, where we can actually draw pictures of what we're doing and make some intelligent guesses as to what has gone wrong.

The Interpolation Problem in 1D

One motivation for this talk is to try to counter the idea that interpolation is simple, and that, given an arbitrary set of m-dimensional data values $\{x_i\}$, that it is a simple matter to "invert the interpolation map" and thus produce a function $g(x)$ that interpolates the data.

Even in 1D, we can easily encounter interpolation problems that will cause severe problems if not handled correctly.

The Interpolation Problem in 1D

A second motivation is to look at the strengths and limitations of polynomial interpolation.

You may be familiar with the idea that a polynomial can't approximate a step function, and that a sequence of polynomial interpolants may seem to be doing a better and better job of approximation until some kind of instability appears, after which the results deteriorate violently.

Since polynomial interpolation has many advantages, we'd like to ask whether we can deal with some of its limitations.

The Interpolation Problem in 1D

A third motivation is to emphasize that the interpolation process involves a lot of choices.

To say that a function $g(x)$ interpolates our data $\{x_i, y_i\}$ really doesn't say very much about $g(x)$ at all!

We generally have a lot of choices about what space to select $g(x)$ from, and usually interpolation is only an intermediate goal to some other quantity of interest to us, which might be the integral, the maximum, the average, and so on.

The Interpolation Problem in 1D

Even though this discussion is limited to the 1D problem, it should suggest to you that

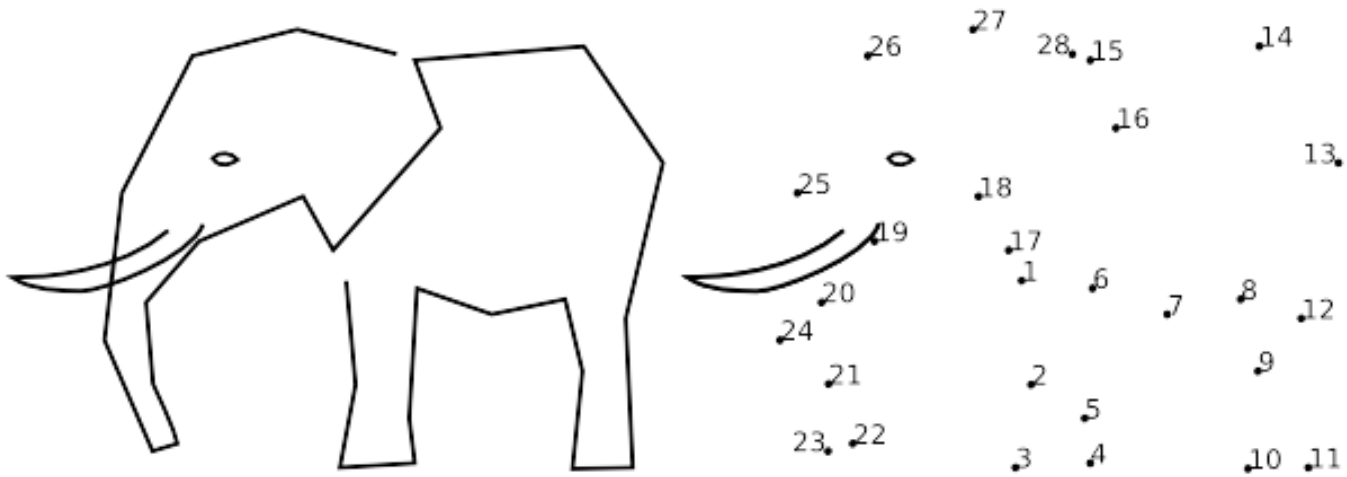
- bad data locations $\{x_i\}$ can doom the process;
- a bad basis for the interpolating space can cause unbounded error;
- a function may interpolate your data but do almost anything elsewhere;
- polynomials can, actually, do a good interpolation job, even for high degree.

Because we can draw pictures of the 1D interpolation problem, we can see and understand some of the issues that will also arise (but invisibly!) in the higher-dimensional problem.

The Interpolation Problem in 1D

The interpolation problem can be thought of as a kind of "connect the dots" puzzle.

However, behind this simple puzzle, there are a variety of hidden choices, expectations, and pitfalls, especially when we think about such problems in a higher dimensional space!



The Interpolation Problem in 1D

- Introduction
- **Interpolation in 1D**
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

INT1D: Interpolation for a 1D Argument

The simplest interpolation problem arise when we wish to represent a process in the form $y = f(x)$, where x is a 1-dimensional input argument, and y is the 1-dimensional observable output.

The actual process $f(x)$ is usually unavailable to us. We think of $f(x)$ as a function, but it might not even give us repeatable results for the same input, or there may be a known statistical error associated with the outputs.

Our task is to look at a finite set of pairs (x_i, y_i), and make some useful model of the process $f(x)$.

INT1D: Prescribed Data

Interpolation of data at prescribed points in 1D:

- Given: a set of m pairs of data (x_i, y_i);
- Find: a suitable function $g(x)$ so that $g(x_i) = y_i$ for all i.

This case can arise when a fixed set of data from experiments is delivered and a representation is desired.

Given that the data set is fixed, it may not be possible to check a model by asking for more data. We might instead drop each data point successively and see how well the reduced data set can "predict" the missing point.

INT1D: Selectable Data

Interpolation of data at selectable points in 1D:

- Given: a procedure or function $f(x)$;
- Find: suitable m, query nodes x_i, and a function $g(x)$ so that $g(x_i) = f(x_i)$ for all i.

This case can arise when a function is available which produces output, but the underlying function is not well understood, or is expensive to evaluate.

Constructing an interpolant can reveal hidden structure in the unknown function, or might be expected to approximate $f(x)$ at a much lower cost (a "surrogate function").

INT1D: Two Kinds of Errors (at least!)

If $g(x)$ is a proposed interpolant to our data, the interpolation error $e = \sum_{i=1}^{n} ||g(x_i) - f(x_i)||$ should be zero.

However, matching the interpolated data is only the beginning. We usually construct an interpolant hoping to approximate $f(x)$ itself over a continuous interval. Thus, we really want to control the L2 error:

$$e = \sqrt{\int_a^b (g(x) - f(x))^2 \quad dx}$$

In test cases, where we know the function $f(x)$, we can easily display the approximation error by plotting $g(x)$ and $f(x)$.

INT1D: A Discrete Measure

Another useful measure is the estimated arc length.

If we have a set of data specified, we can define the piecewise linear interpolant to that data, and measure its arc length.

If we now produce an interpolant to the same data, we can simply evaluate it at, say, 1000 equally spaced points, estimate its arc length, and compare it to the piecewise linear result.

This is a simple way to detect computationally whether the computed interpolant is oscillating between the data.

This check can be extended to 2D and 3D problems in a fairly simple way.

INT1D: Approximation for a 1D Argument

Occasionally, our approach to the interpolation problem breaks down. It may still be possible to produce a model of the underlying function $f(x)$, but perhaps one that only approximates the observed data.

In that case, we would be solving the approximation problem, which seeks a function $g(x)$ which minimizes the pointwise approximation error $e = \sum_{i=1}^{n} ||g(x_i) - f(x_i)||$ to our data.

In this case, we give up the interpolation property in the hope that the resulting function $g(x)$ can be made smoother, or simpler than an interpolant.

The most common example of this approach is the use of the least-squares polynomial.

INT1D: An Unrealistic Wish List

Desirable properties for an interpolant to n data values:

- continuity: small δx means small $\delta g(x)$;
- differentiability: small δx means linear $\delta g(x)$;
- monotonicity: if data is strictly rising, so is $g(x)$;
- convexity: $g(x)$ lies between data values;
- non-oscillation: no excessive "wiggles";
- robustness: small errors in x_i or y_i don't matter much;
- locality: nearby data is most important;
- cheap to define: cost $\mathcal{O}(n)$, $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$?
- cheap to evaluate: cost $\mathcal{O}(n)$, $\mathcal{O}(n^2)$, $\mathcal{O}(n^3)$?
- extensible: can we add more data values easily?
- analytic: can we estimate errors?
- adaptable: can we reduce errors that we detect?

INT1D: Choosing an Interpolation Space

A given set of data has many possible interpolants.

We want to frame the problem in such a way that the answer $g(x)$ is computable, has some properties from our wish list, and is a good model of the underlying function $f(x)$.

We describe our problem as determining the unique element $g(x)$ from some appropriate function space $\mathbb{G}$ which is to be selected because it interpolates our data.

For example, a common choice for $\mathbb{G}$ is $\mathbb{P}^{n-1}$, the space of polynomials of degree $n-1$ (or "order" n).

The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- **Nearest Interpolation**
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

NEAR: The Nearest Neighbor

Almost the simplest interpolatory idea to define the value $g(x)$ is to locate the nearest data location x_i and set $g(x) = y_i$.

Let $\phi_i(x)$ be the characteristic function of the interval $[a_i, b_i]$ of points closest to data location x_i:

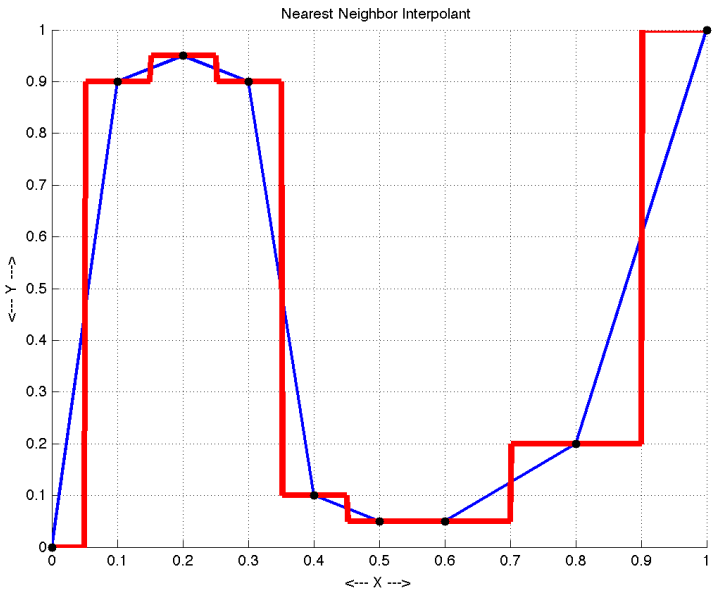
$$\phi_i(x) = \begin{cases} 1 \text{ if } x \in [a_i, b_i]; \\ 0 \text{ otherwise;} \end{cases}$$

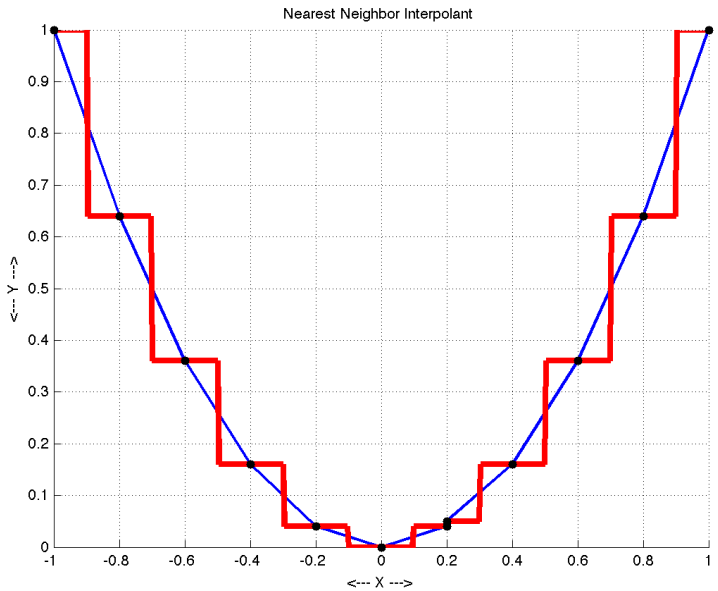
Then our interpolant is

$$g(x) = \sum_{i=1}^{n} y_i \phi_i(x)$$

We will see this kind of representation of the interpolant repeatedly. The $\phi(x)$ functions act as a basis for $\mathcal{G}$, and the numbers y_i play the role of coordinates in that basis.

NEAR: Example Interpolants





NEAR: The Nearest Neighbor

Nearest neighbor interpolation guarantees locality, - extremely so.

However, the interpolant will not even be continuous.

No model of the relationship is provided; at best, this model is hidden in the arrangement of the data.

A more serious problem is that, to get locality, we have to examine the entire set of data. Each evaluation of $g(x)$ requires computing the distance to every data point. In 1D we could sort the data, in 2D or 3D, we could use triangulation; after that, this becomes a significant cost.

The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- **Piecewise Linear Interpolation**
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

PWL: Piecewise Linear Interpolation

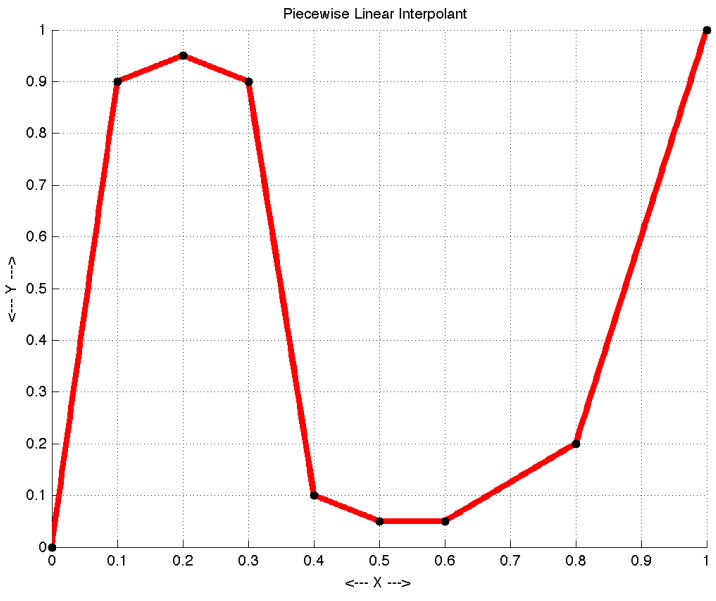
A powerful and simple interpolatory idea combines the appeal of linear functions with the caution of locality; we define the interpolant as a piecewise function which is linear between successive data locations.

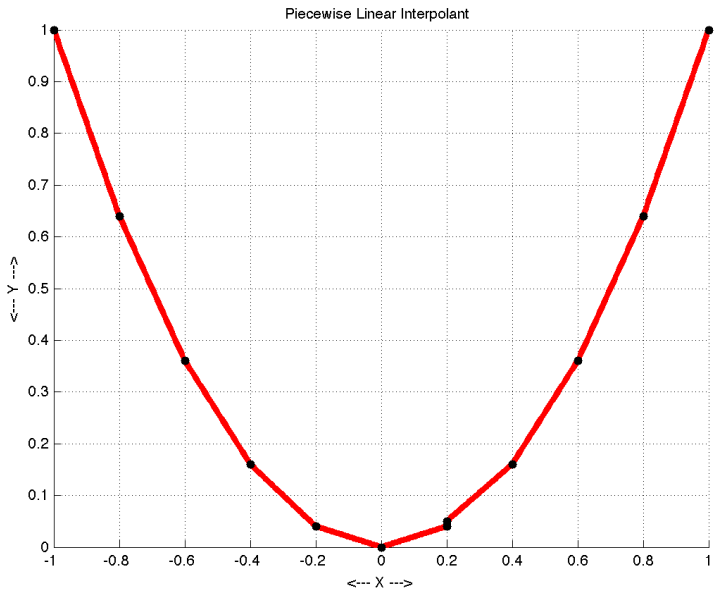
Let $\phi_i(x)$ be the function which linearly rises from 0 to 1 over $[x_{i-1}, x_i]$, drops from 1 to 0 over $[x_i, x_{i+1}]$, and is 0 elsewhere. Then our interpolant is

$$g(x) = \sum_{i=1}^{n} y_i \phi_i(x)$$

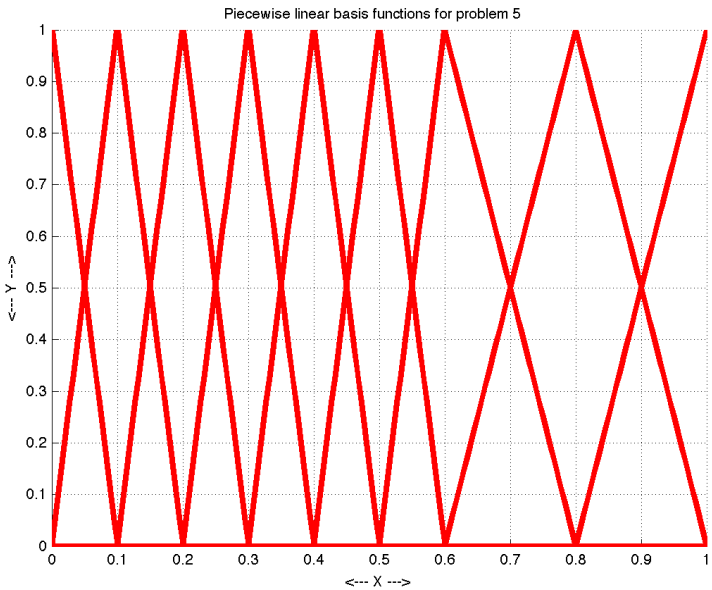
which again gives us the basis for the underlying space $\mathcal{G}$, and an explicit formula for the interpolant in terms of the data.

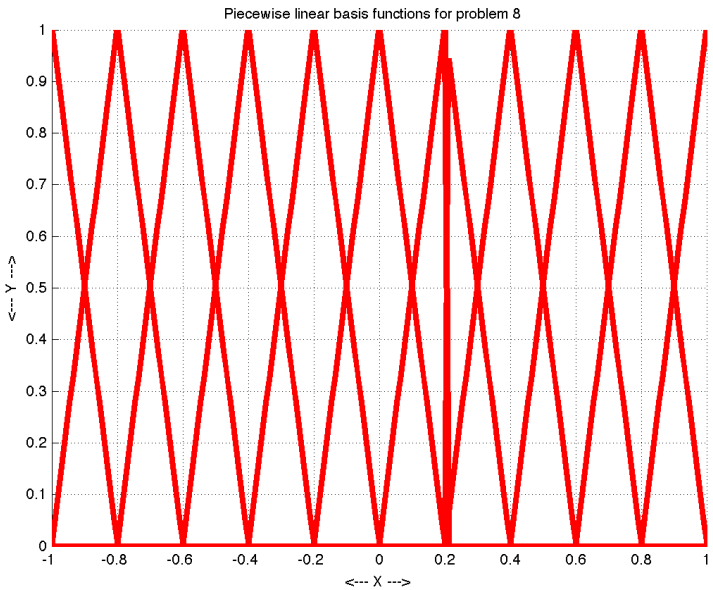
PWL: Example Interpolants





PWL: Example Basis Functions





PWL: The Piecewise Linear Interpolant

PWL interpolation combines locality, continuity, and linearity (almost).

In the 1D case, we need to sort the data locations before processing, in order to create the subintervals $[x_i, x_{i+1}]$, although we often don't even think about this, and the data is typically already sorted for us.

Extending this idea to higher regions, a greater effort is necessary in order to "sort" the data locations: in 2D, we need local triangles, in 3D tetrahedrons, and after that simplices, whose creation becomes a significant cost.

A strength of PWL interpolation is that it is easy to locally adapt the interpolant. Adding a new data location simply means we have to replace one interval by two smaller ones.

The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- **Vandermonde Interpolation**
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

VANI: The Vandermonde Interpolant

A straightforward way to approach the problem of interpolating m sets of data values is to assume a form for the interpolating function that will have exactly m degrees of freedom c.

Our interpolation conditions will be:

$$f(c; x_1) = y_1$$
$$f(c; x_2) = y_2$$
$$\ldots$$
$$f(c; x_m) = y_m$$

A natural choice is a polynomial $p(x)$ of degree $m - 1$,

$$p(x) = c_0 + c_1 x + c_2 x^2 + \ldots + c_{m-1} x^{m-1}$$

VANI: The Vandermonde Interpolant

Our interpolation conditions become

$$
\begin{aligned}
c_0 * 1 + c_1 * x_1 + c_2 * x_1^2 + \ldots + c_{m-1} * x_1^{m-1} &= y_1 \\
c_0 * 1 + c_1 * x_2 + c_2 * x_2^2 + \ldots + c_{m-1} * x_2^{m-1} &= y_2 \\
&\ldots \\
c_0 * 1 + c_1 * x_m + c_2 * x_m^2 + \ldots + c_{m-1} * x_m^{m-1} &= y_m
\end{aligned}
$$

or $A * c = y$ where A is the Vandermonde matrix.

We have only to invert the matrix A to solve for the coefficients c and hence recover our interpolant $p(x)$.

This is one example of "inverting the interpolation map".

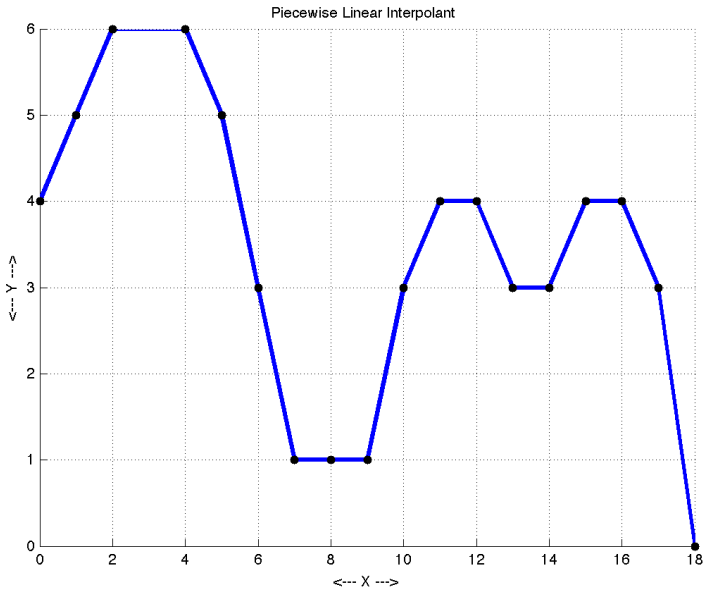
VANI: Vandermonde Approach is Problematic

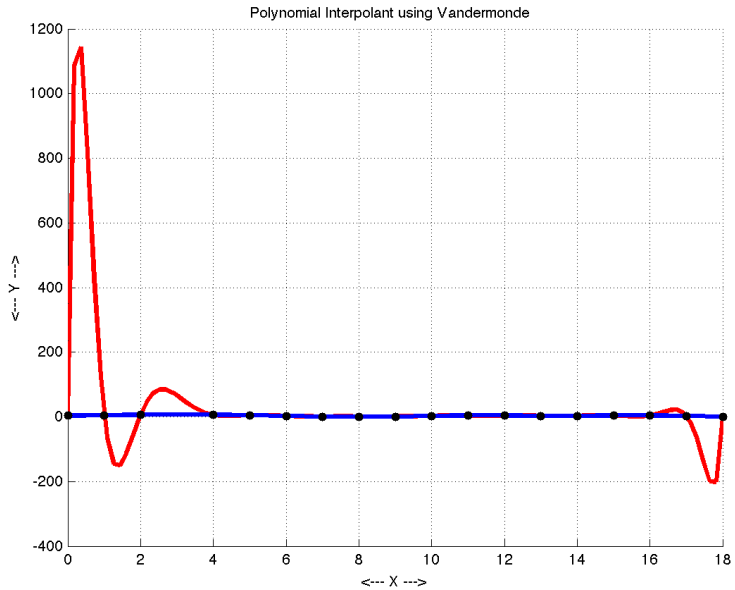
It is known that, if the nodes x_i are distinct, then the matrix A is nonsingular, the system is solvable for c, and hence the polynomial interpolant $p(x)$ can be recovered automatically from the nodes x_i and the data y_i.

The Vandermonde matrix may be nonsingular, but in practice it becomes very poorly conditioned with increasing m:

- systems with small m can be accurately solved;
- systems with moderate m are solved with severe errors;
- systems with large m are numerically singular and cause linear solvers to fail *(you can't even compute a bad solution!)*.

VANI: Vandermonde Experiment 1





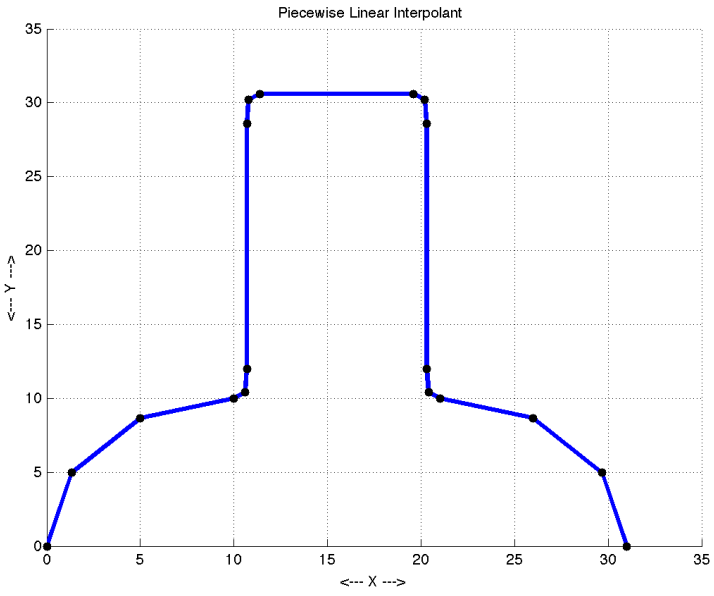
Condition of A $\approx 1.1 \times 10^{23}$

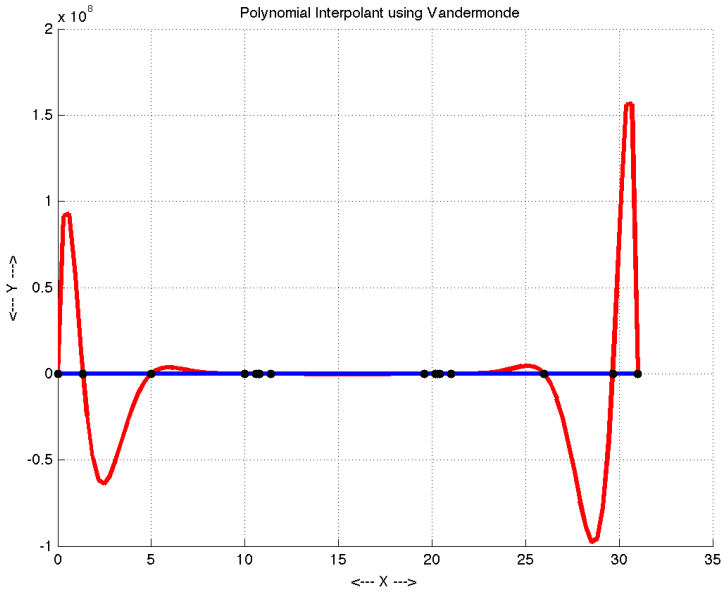
Average interpolation error $= 7.2 \times 10^{-4}$

PW Linear length $= 3.13$ units

Polynomial length $= 559.30$ units.

VANI: Vandermonde Experiment 2





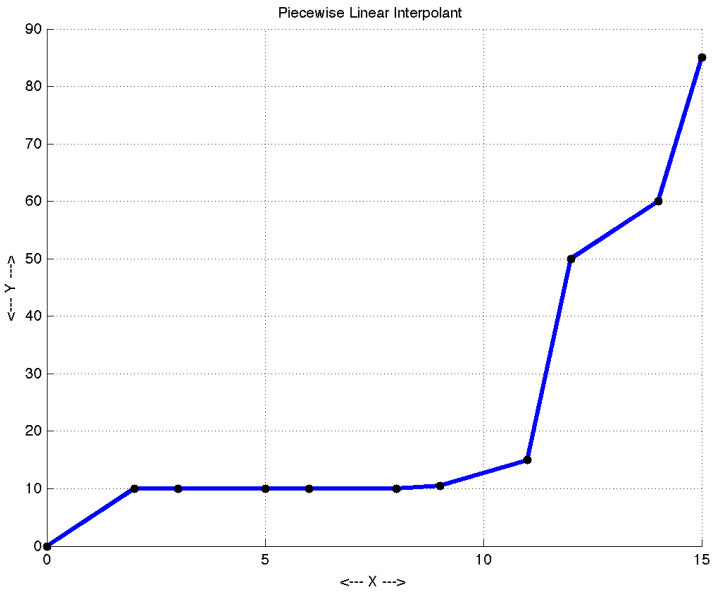
Condition of A $\approx 1.3 \times 10^{27}$

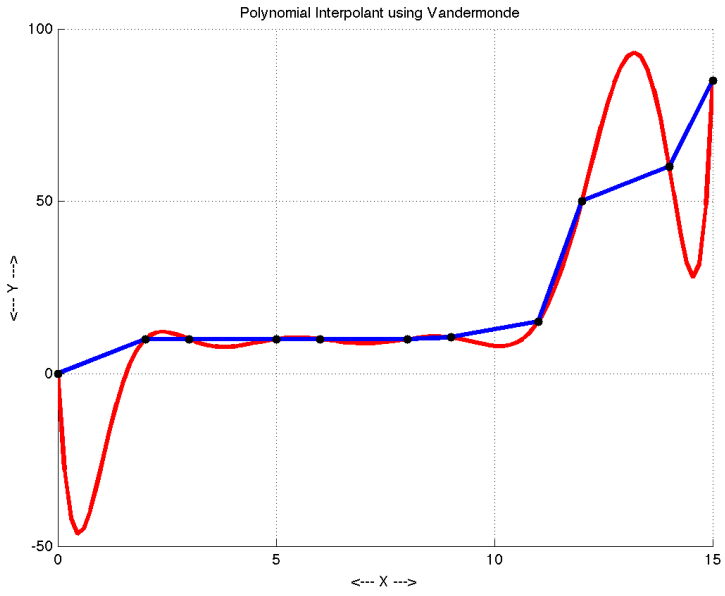
Average interpolation error $= 2.9$

PW Linear length $= 2.66$ units

Polynomial length $= 2.9 \times 10^{27}$ units.

VANI: Vandermonde Experiment 3





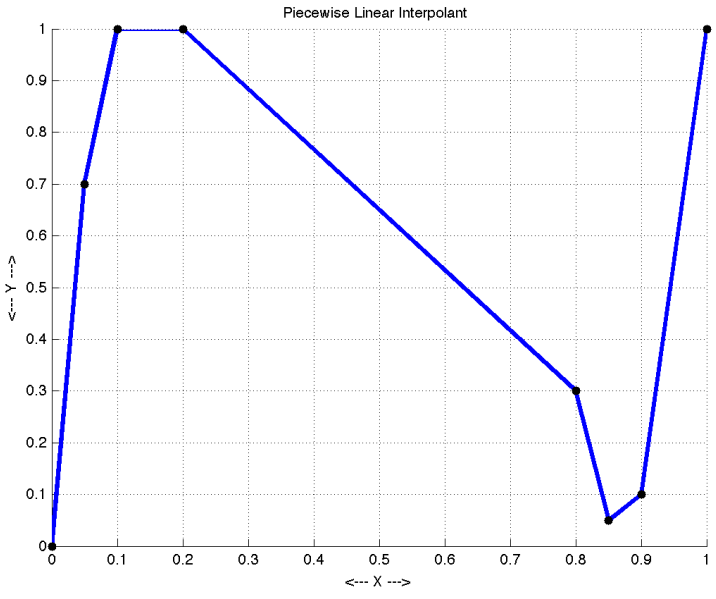
Condition of A $\approx 1.6 \times 10^{14}$

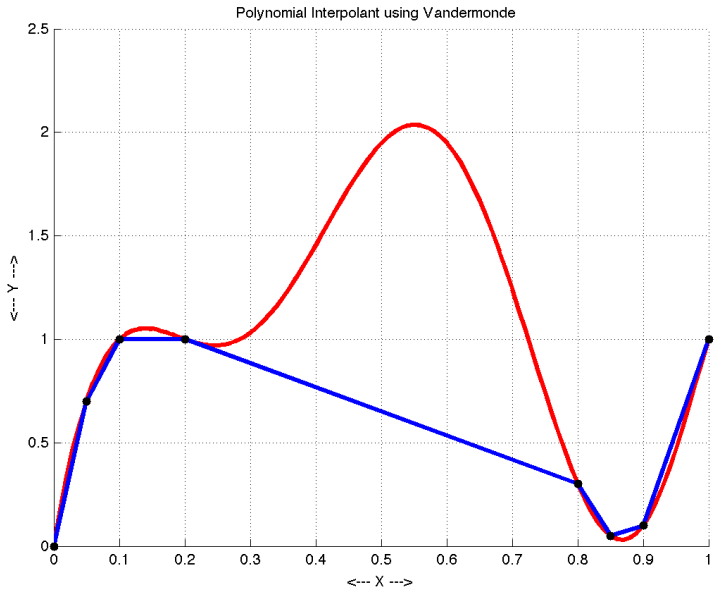
Average interpolation error $= 1.4 \times 10^{-09}$

PW Linear length $= 3.13$ units

Polynomial length $= 559.30$ units.

VANI: Vandermonde Experiment 4





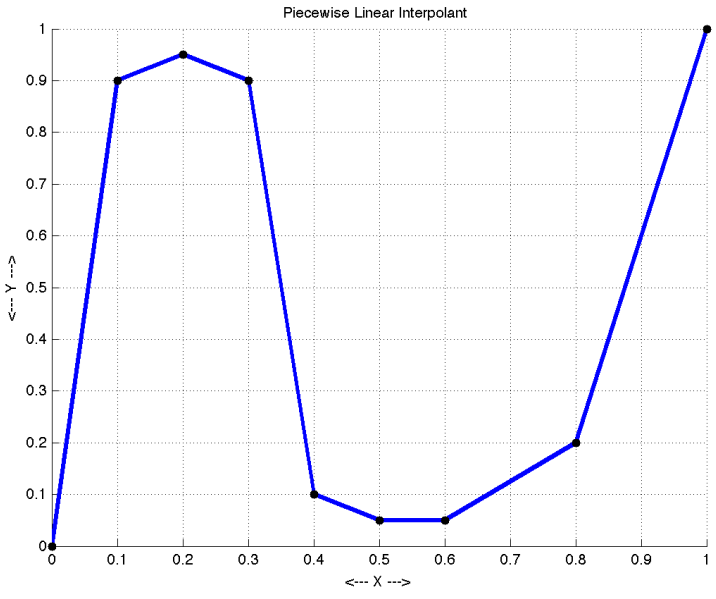
Condition of A ≈ 356159

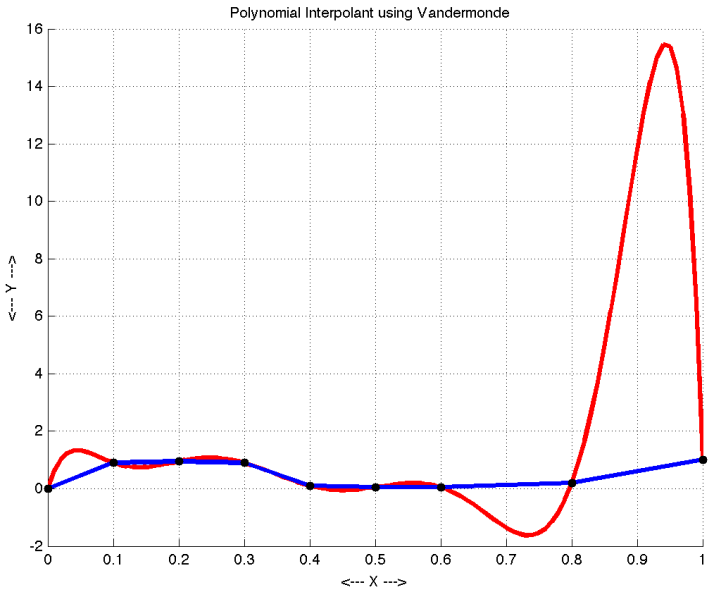
Average interpolation error $= 2.7 \times 10^{-14}$

PW Linear length $= 3.25$ units

Polynomial length $= 5.38$ units.

VANI: Vandermonde Experiment 5





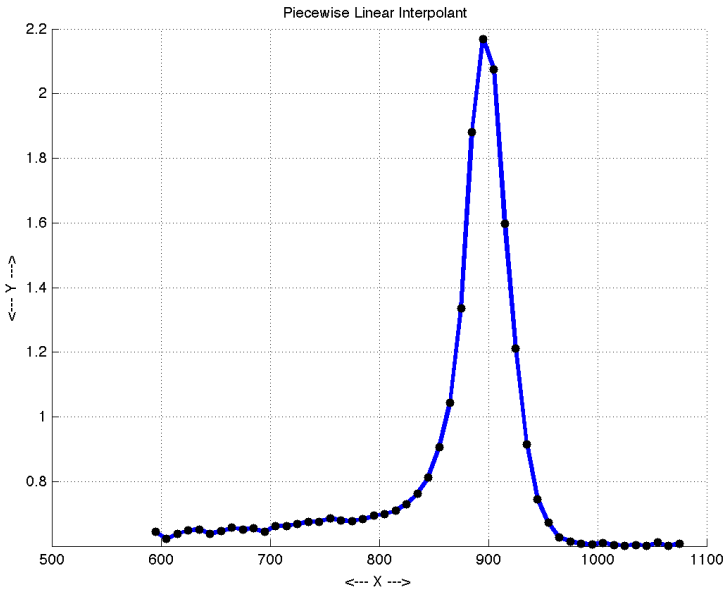
Condition of A $\approx 4.1 \times 10^6$

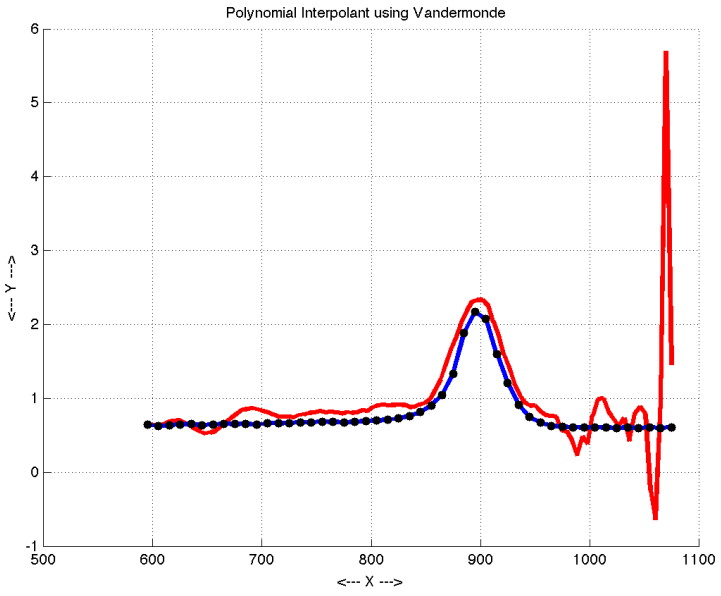
Average interpolation error $= 2.3 \times 10^{-12}$

PW Linear length $= 3.22$ units

Polynomial length $= 37.19$ units.

VANI: Vandermonde Experiment 6





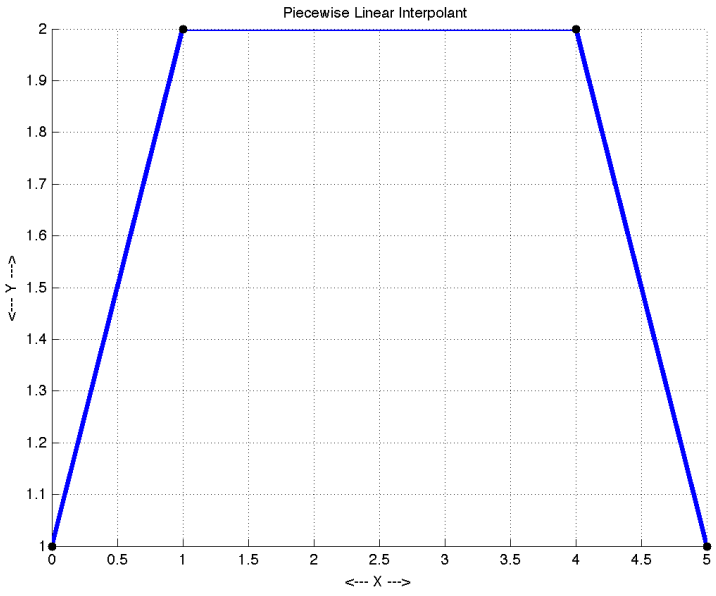
Condition of A $\approx 2 \times 10^{146}$

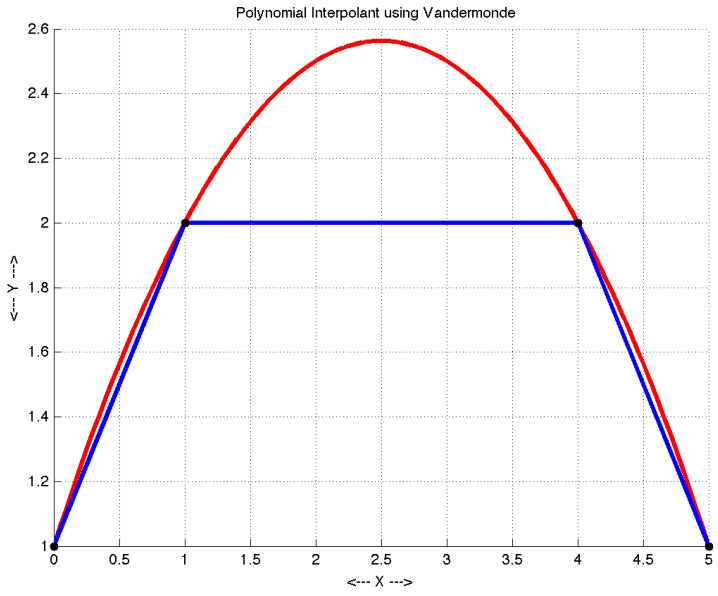
Average interpolation error $= 3.6 \times 10^{-2}$

PW Linear length $= 1.68$ units

Polynomial length $= 23.47$ units.

VANI: Vandermonde Experiment 7





Condition of A ≈ 367

Average interpolation error $= 0$

PW Linear length $= 2.63$ units

Polynomial length $= 3.36$ units.

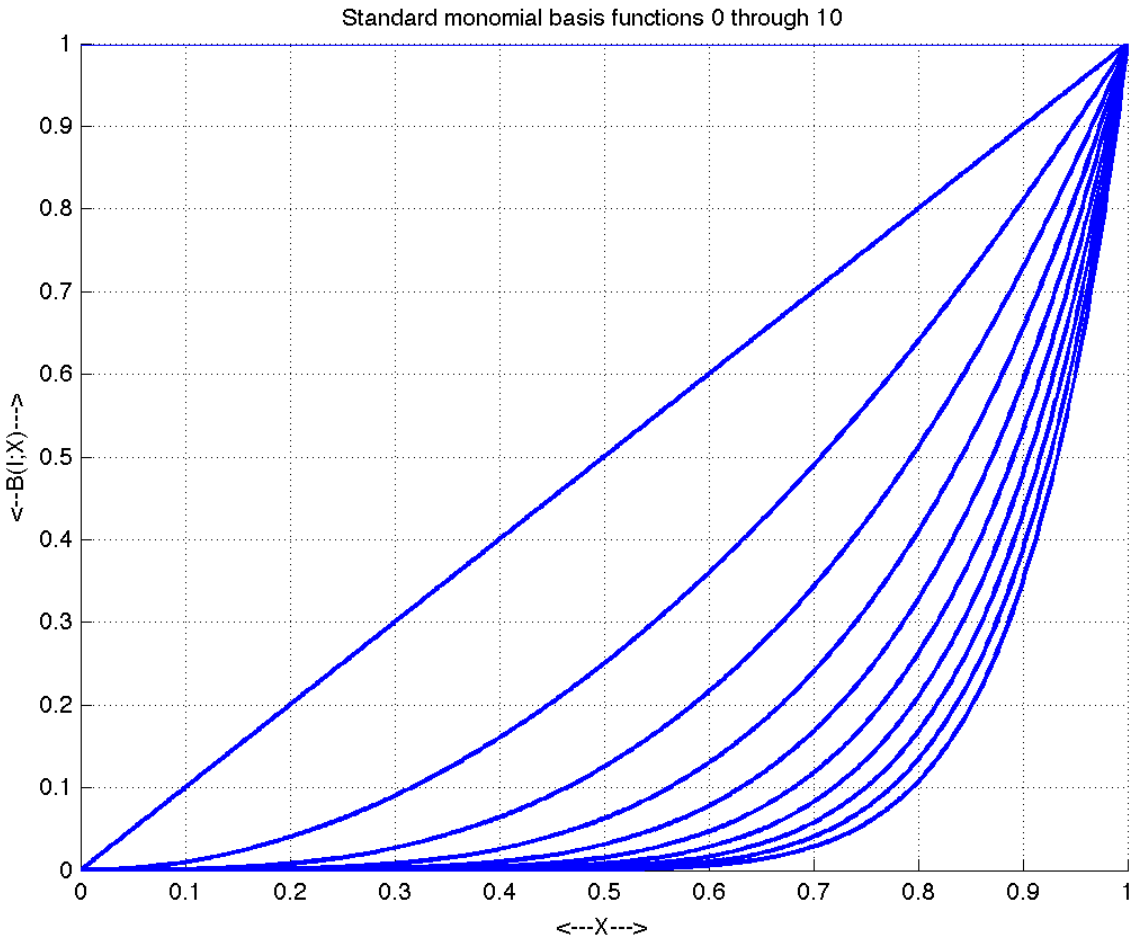
VANI: Vandermonde Basis Functions

Assuming our nodes are equally spaced, the picture of the basis functions can also be easily interpreted as a representation of the matrix entries itself.

Matrix column:

- 1 is 0-degree basis function (all 1's).
- 2 is linear basis functions (0, 1/10, 2/10, ..., 1);
- 3 is quadratic basis functions (0, 1/100, 4/100, ..., 1);

VANI: Vandermonde Basis Functions



VANI: The Vandermonde Matrix

	x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^10
0.0	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.1	1.0000	0.1000	0.0100	0.0010	0.0001	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
0.2	1.0000	0.2000	0.0400	0.0080	0.0016	0.0003	0.0001	0.0000	0.0000	0.0000	0.0000
0.3	1.0000	0.3000	0.0900	0.0270	0.0081	0.0024	0.0007	0.0002	0.0001	0.0000	0.0000
0.4	1.0000	0.4000	0.1600	0.0640	0.0256	0.0102	0.0041	0.0016	0.0007	0.0003	0.0001
0.5	1.0000	0.5000	0.2500	0.1250	0.0625	0.0312	0.0156	0.0078	0.0039	0.0020	0.0010
0.6	1.0000	0.6000	0.3600	0.2160	0.1296	0.0778	0.0467	0.0280	0.0168	0.0101	0.0060
0.7	1.0000	0.7000	0.4900	0.3430	0.2401	0.1681	0.1176	0.0824	0.0576	0.0404	0.0282
0.8	1.0000	0.8000	0.6400	0.5120	0.4096	0.3277	0.2621	0.2097	0.1678	0.1342	0.1074
0.9	1.0000	0.9000	0.8100	0.7290	0.6561	0.5905	0.5314	0.4783	0.4305	0.3874	0.3487
1.0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- **Vandermonde Approximation**
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

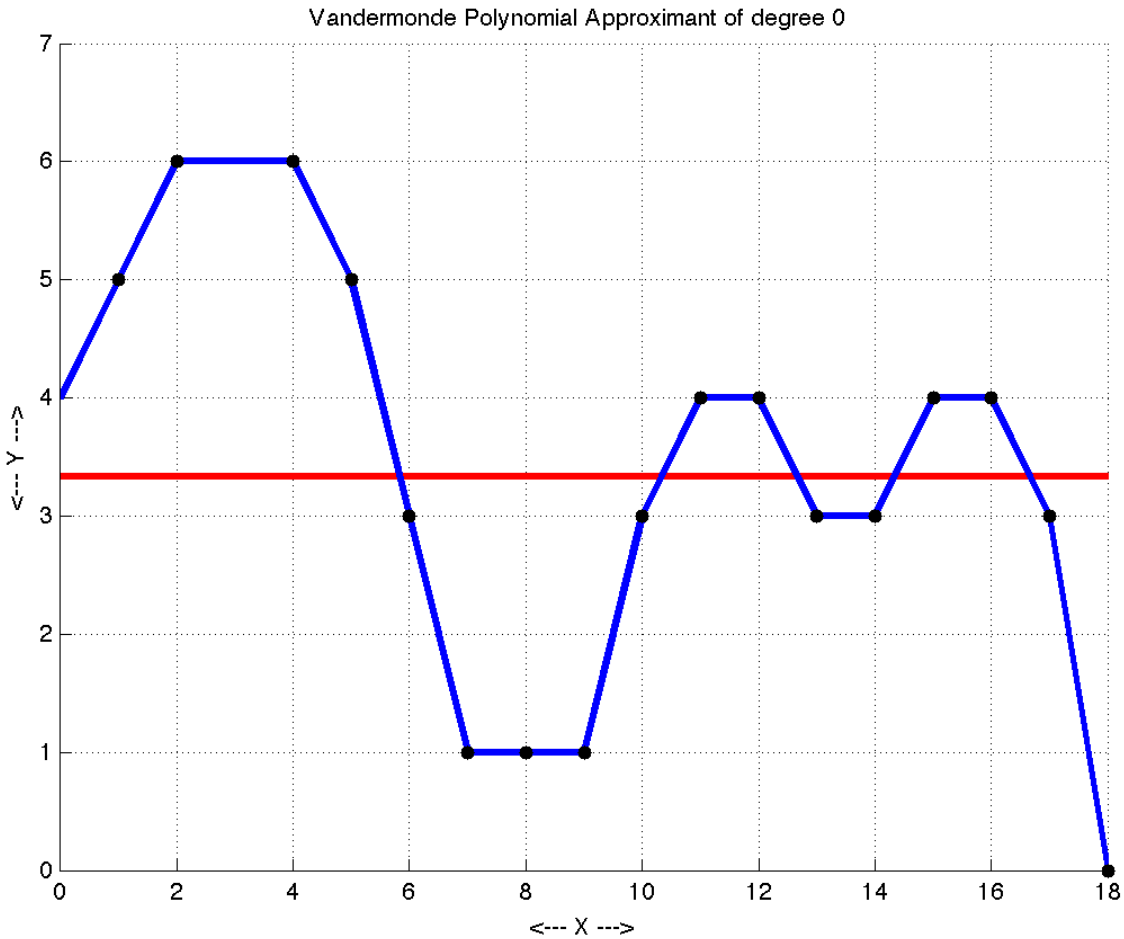
VANA: The Vandermonde Approximant

Since the high value of m is causing the problem, we can try to relax our requirements so that we are interpolating at m points, but using a polynomial of degree $n < m$. Our approximation conditions become

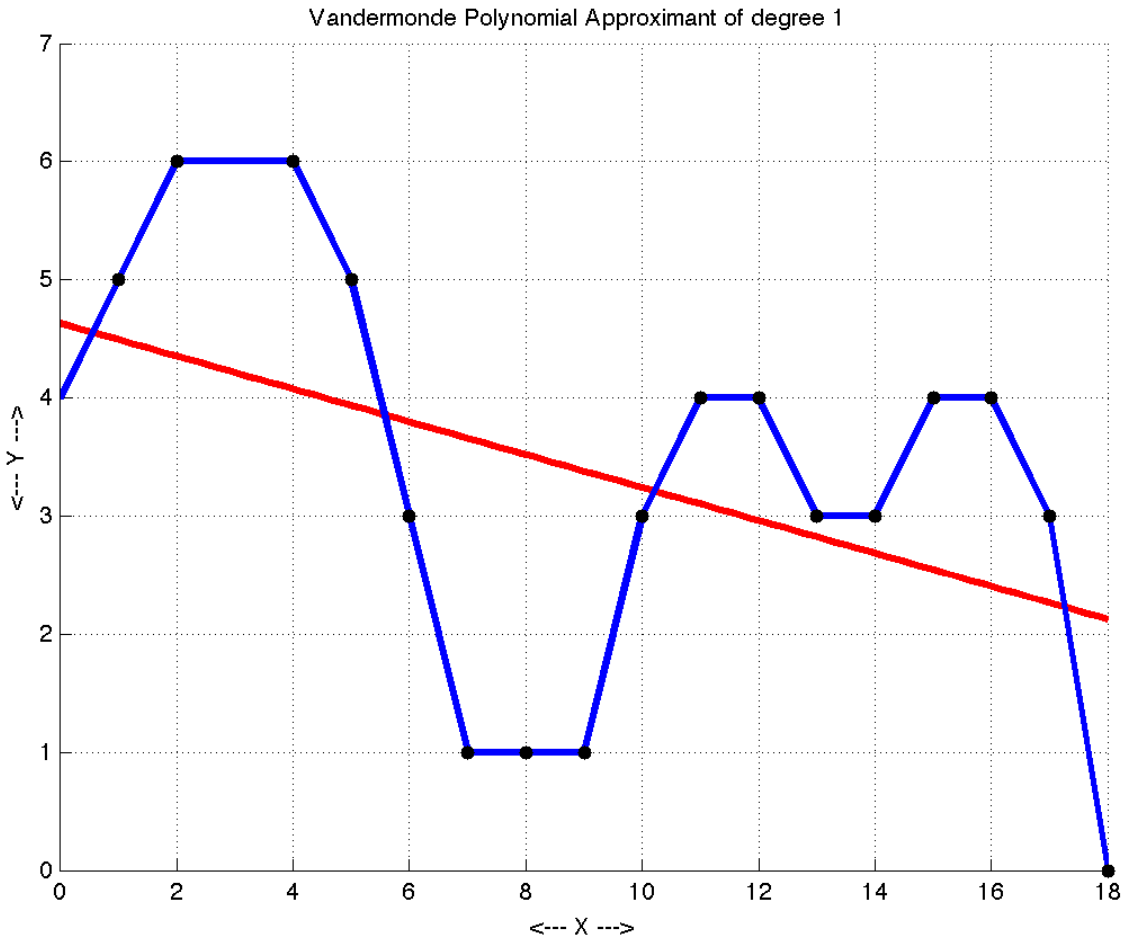
$$c_0 * 1 + c_1 * x_1 + c_2 * x_1^2 + \ldots + c_{n-1} * x_1^{n-1} = y_1$$
$$c_0 * 1 + c_1 * x_2 + c_2 * x_2^2 + \ldots + c_{n-1} * x_2^{n-1} = y_2$$
$$\ldots$$
$$c_0 * 1 + c_1 * x_m + c_2 * x_m^2 + \ldots + c_{n-1} * x_m^{n-1} = y_m$$

and now we have an *mxn* overdetermined linear system $A * c = y$ to solve for the coefficients. As long as the matrix A has full column rank *(uh oh)* we can solve this approximation problem.

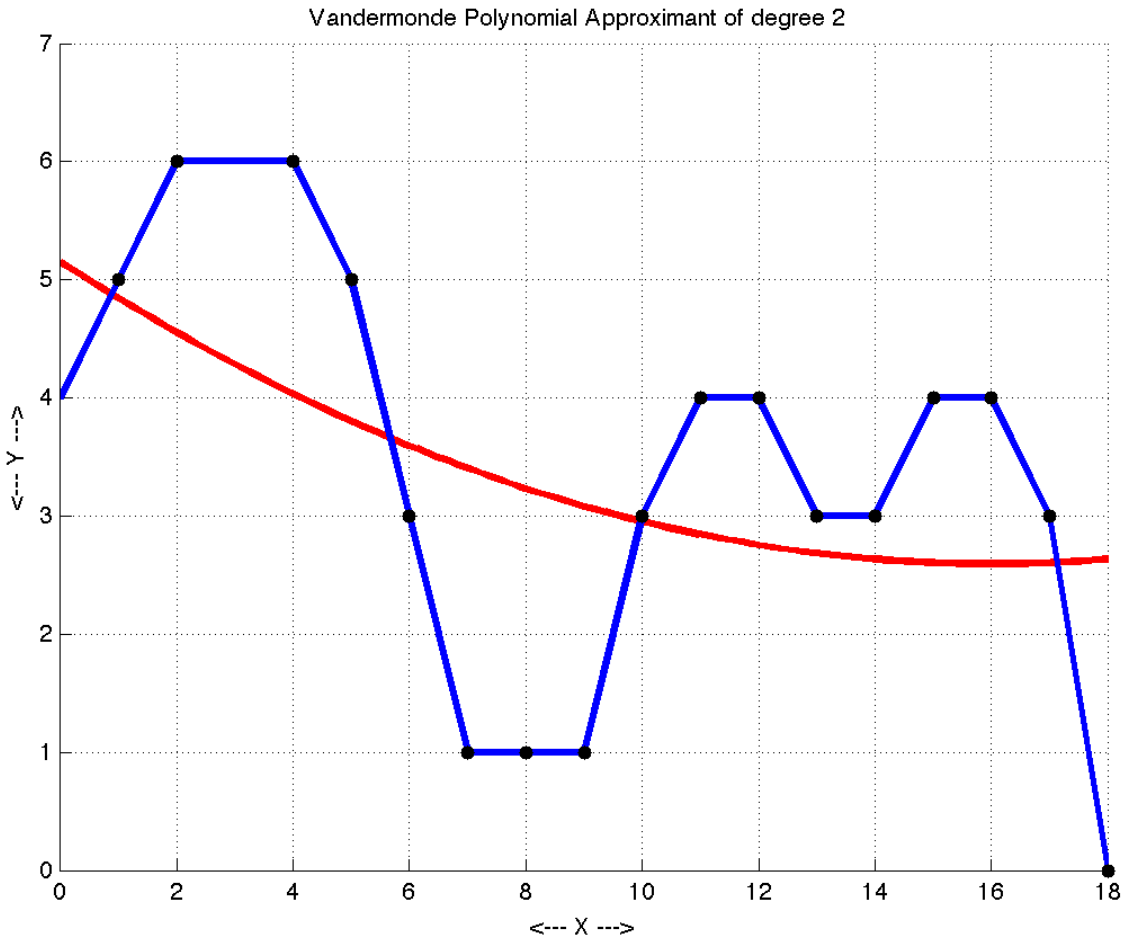
VANA: Vandermonde Approximant N = 0



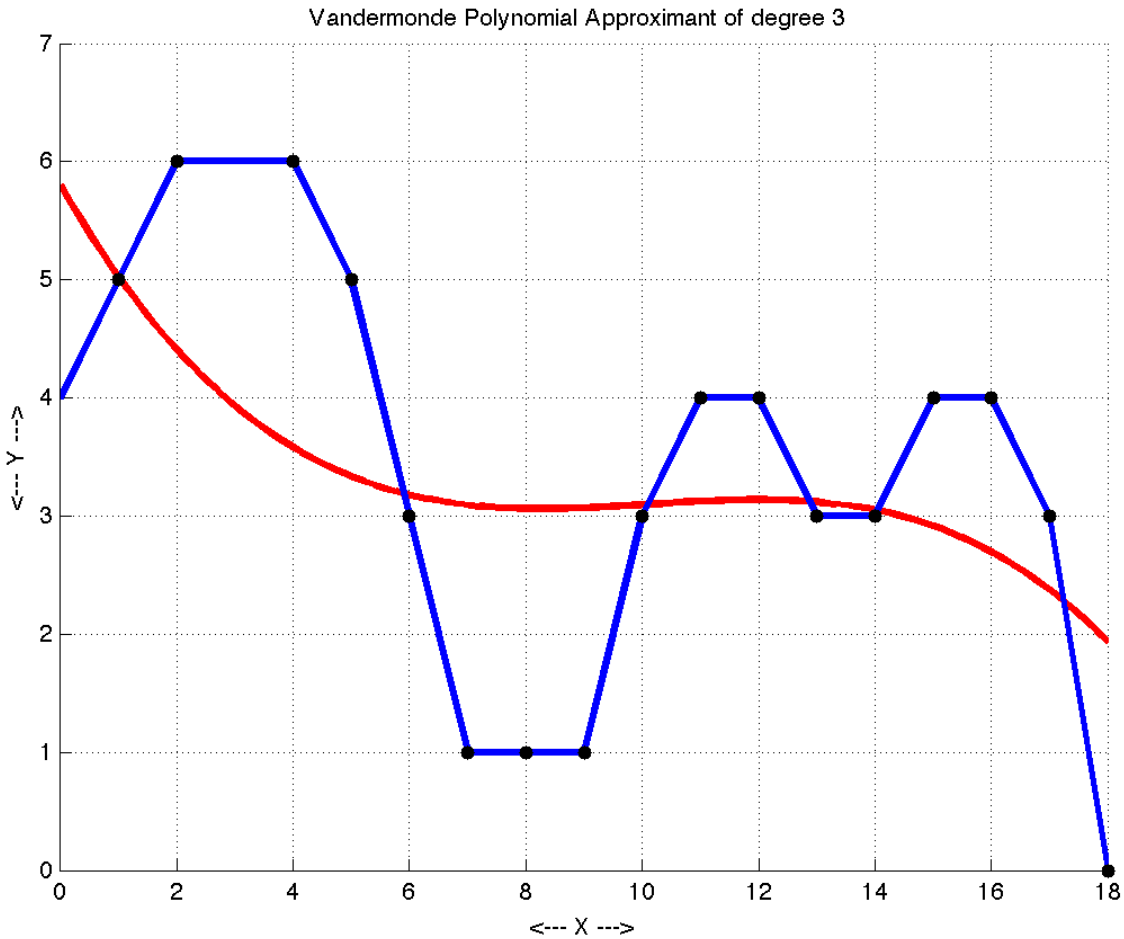
VANA: Vandermonde Approximant N = 1



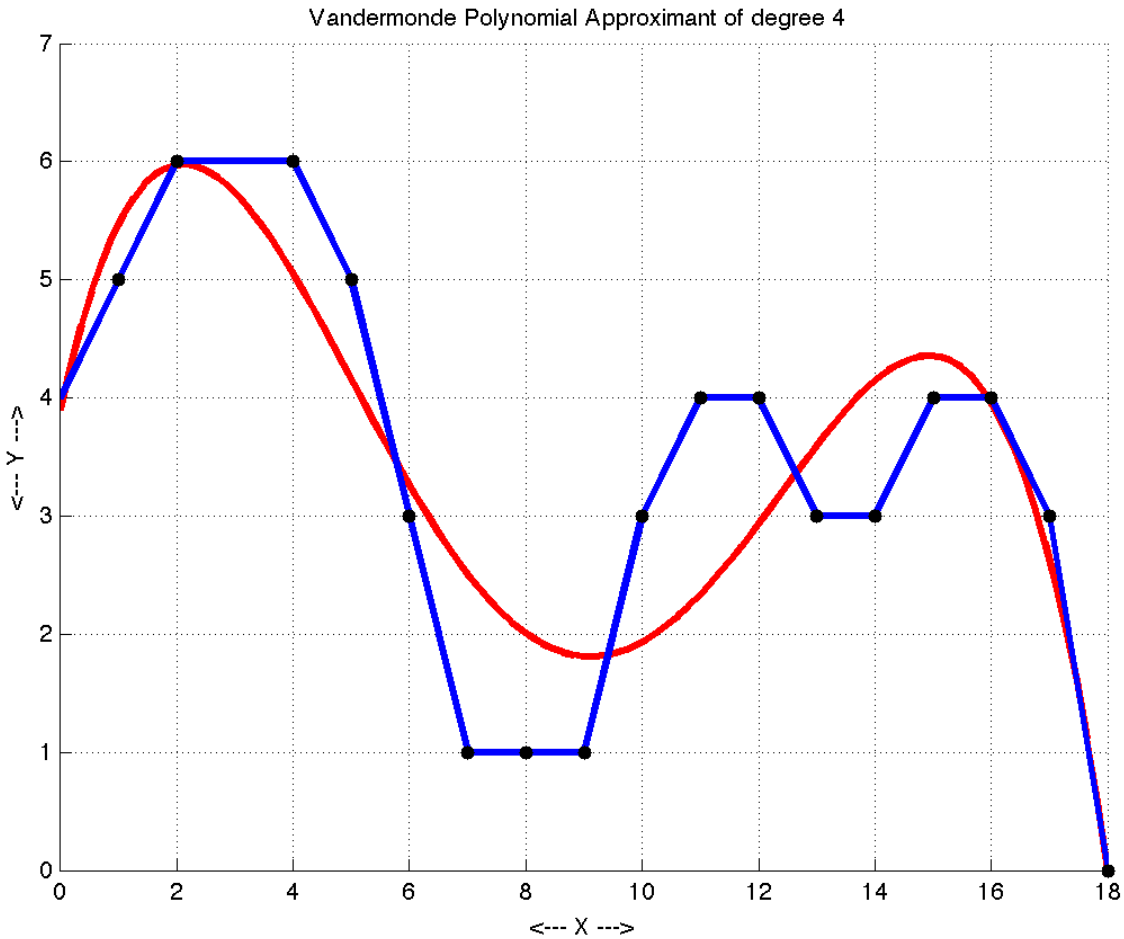
VANA: Vandermonde Approximant N = 2



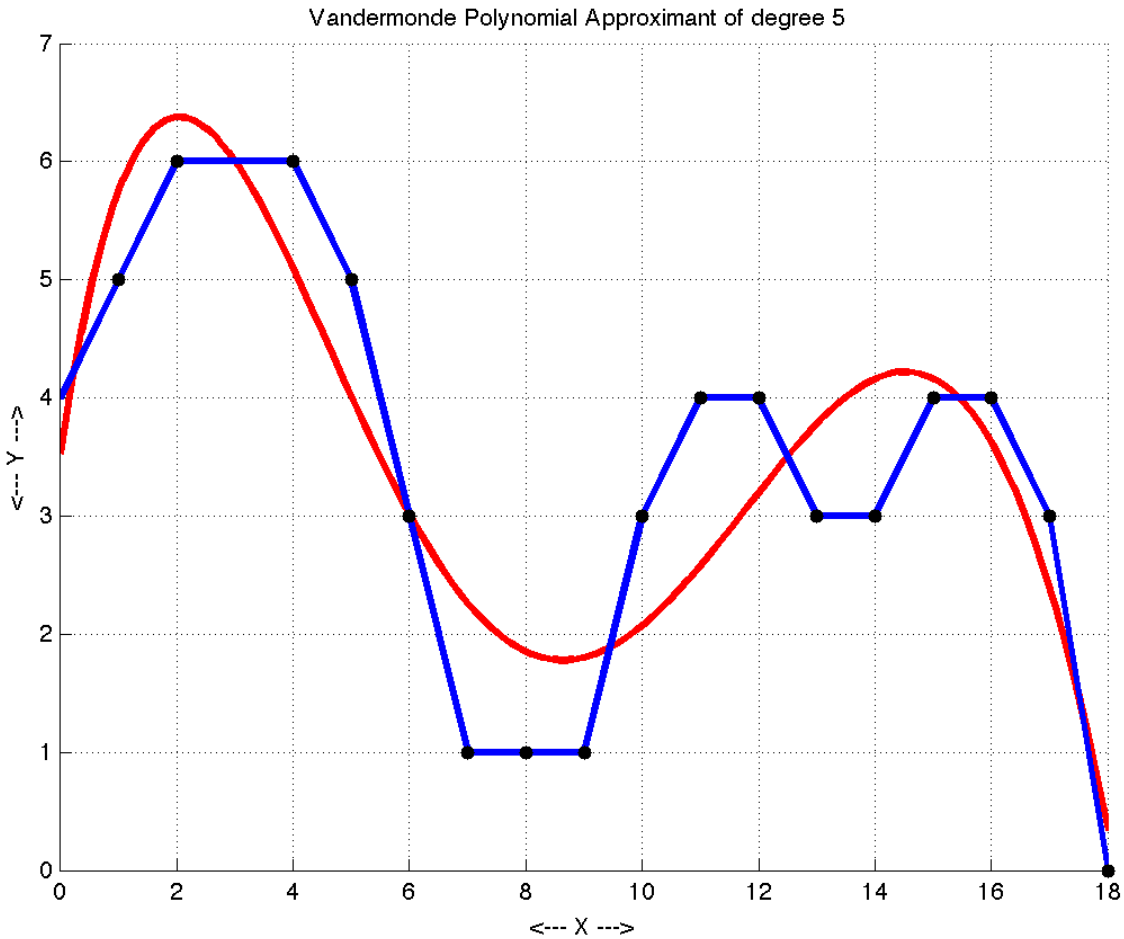
VANA: Vandermonde Approximant N = 3



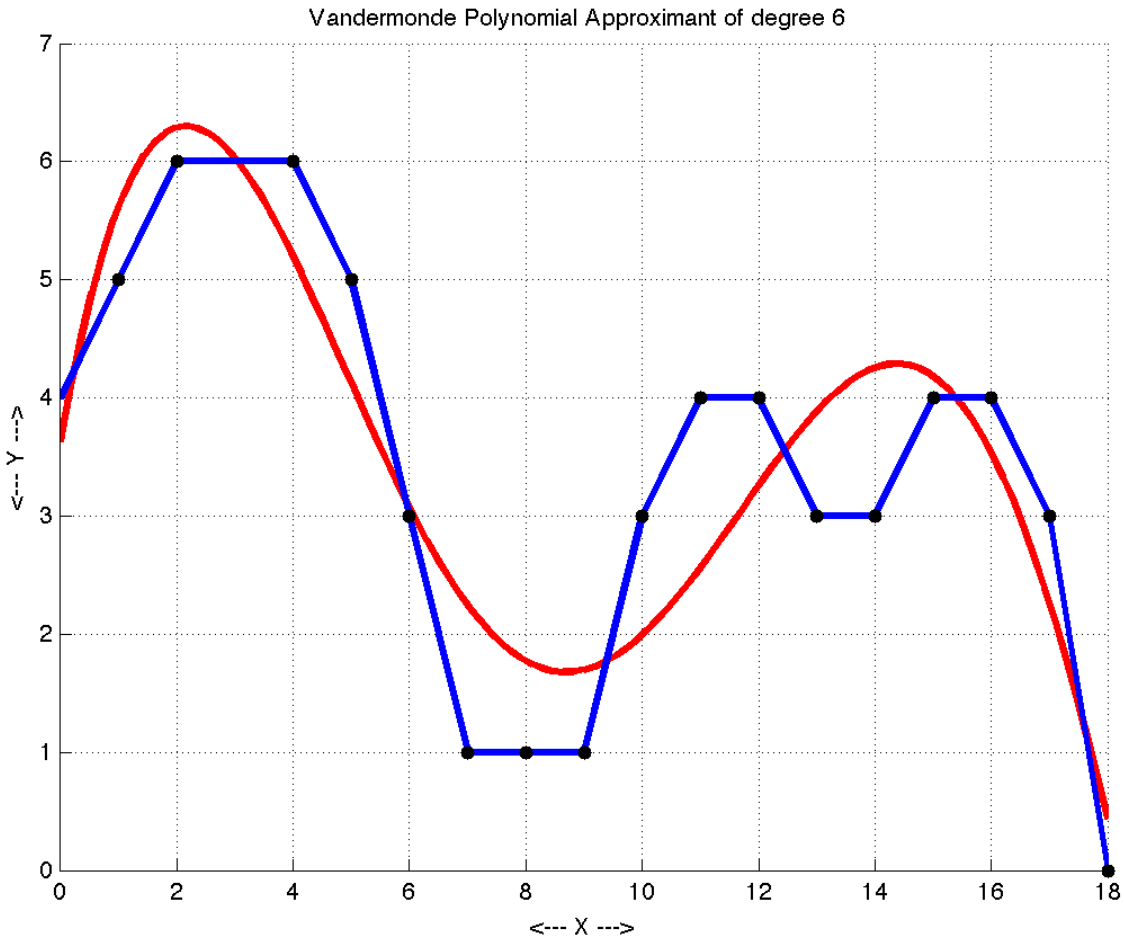
VANA: Vandermonde Approximant N = 4



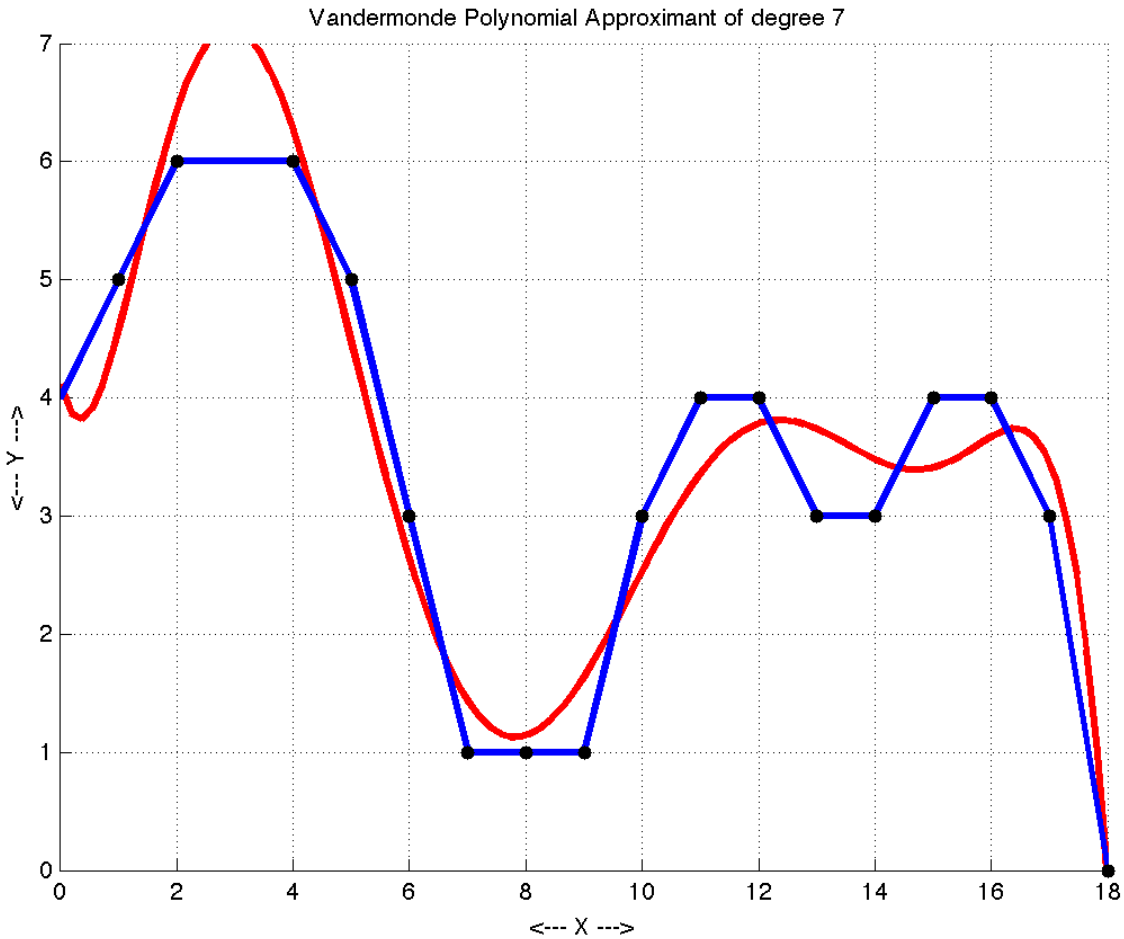
VANA: Vandermonde Approximant N = 5



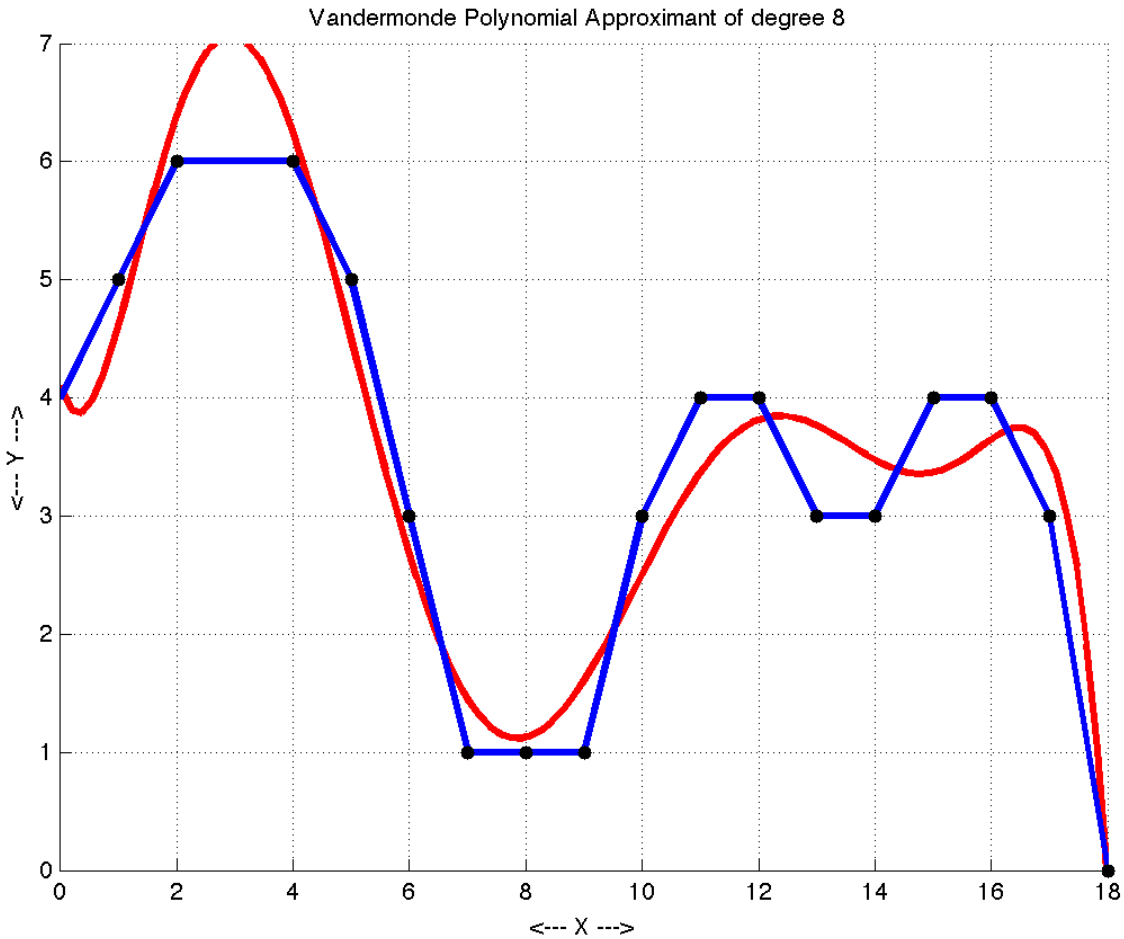
VANA: Vandermonde Approximant N = 6



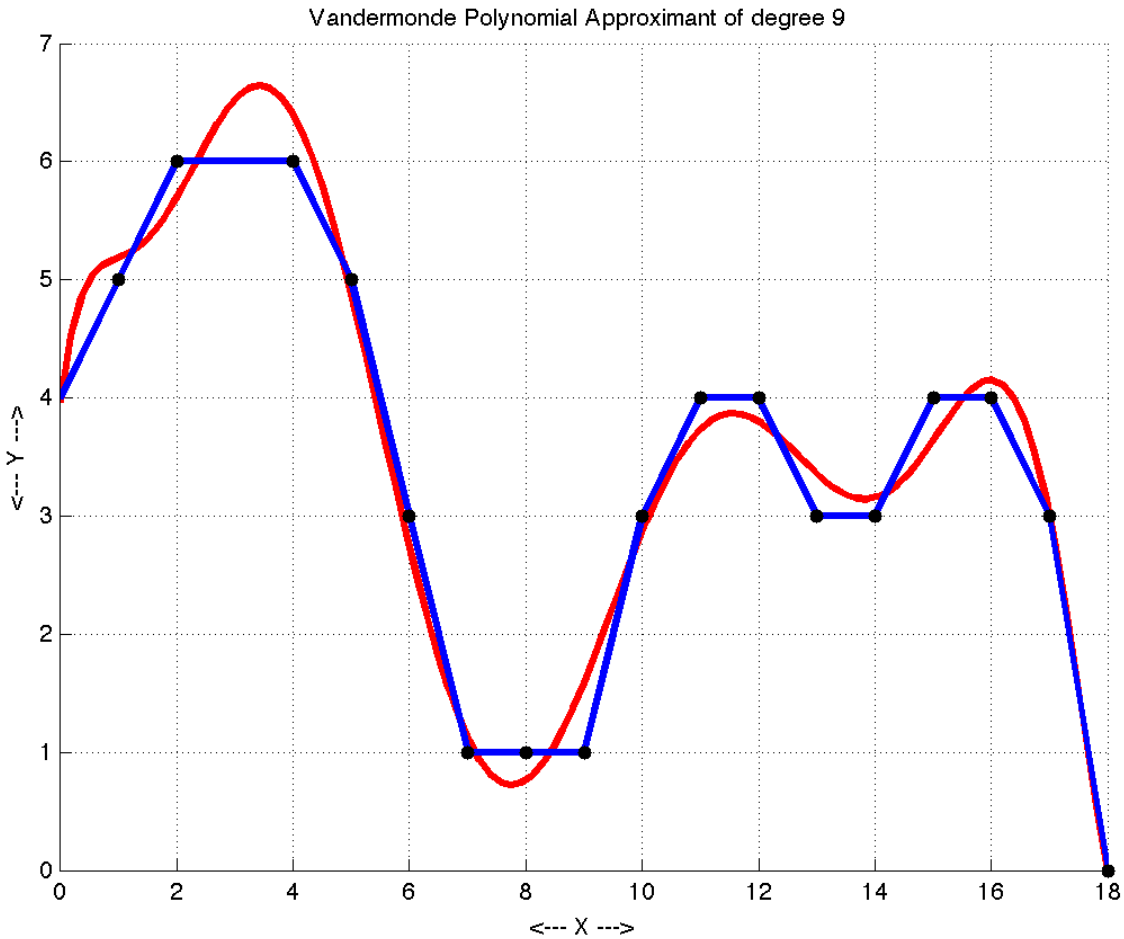
VANA: Vandermonde Approximant N = 7



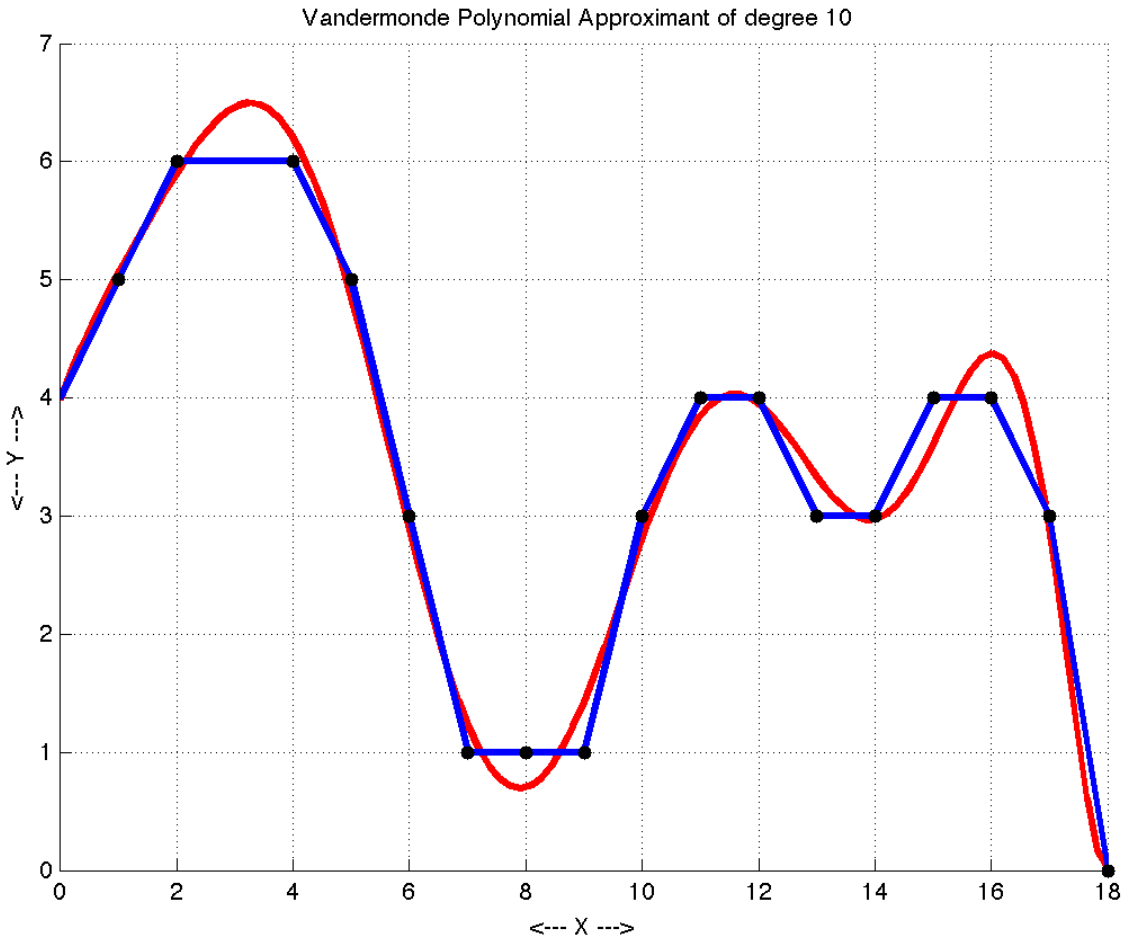
VANA: Vandermonde Approximant N = 8



VANA: Vandermonde Approximant N = 9

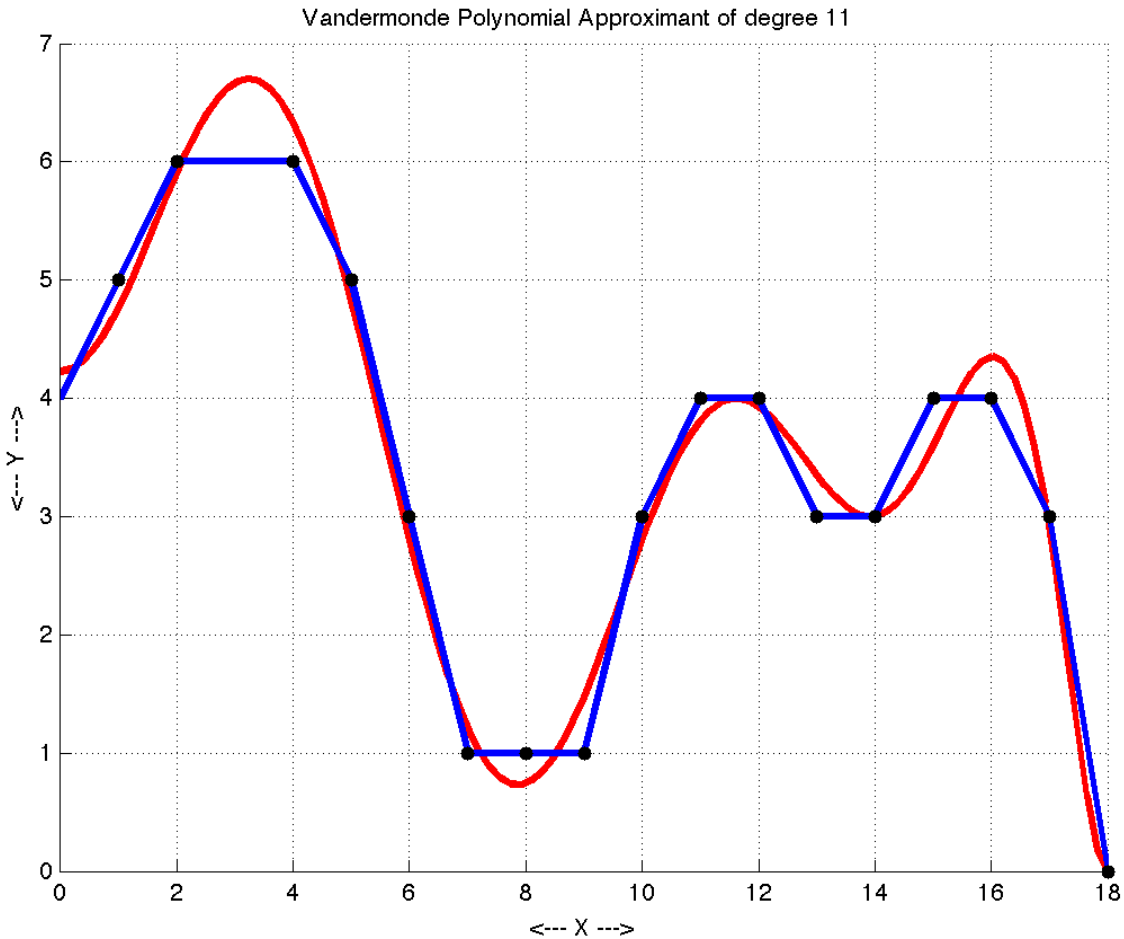


VANA: Vandermonde Approximant N = 10



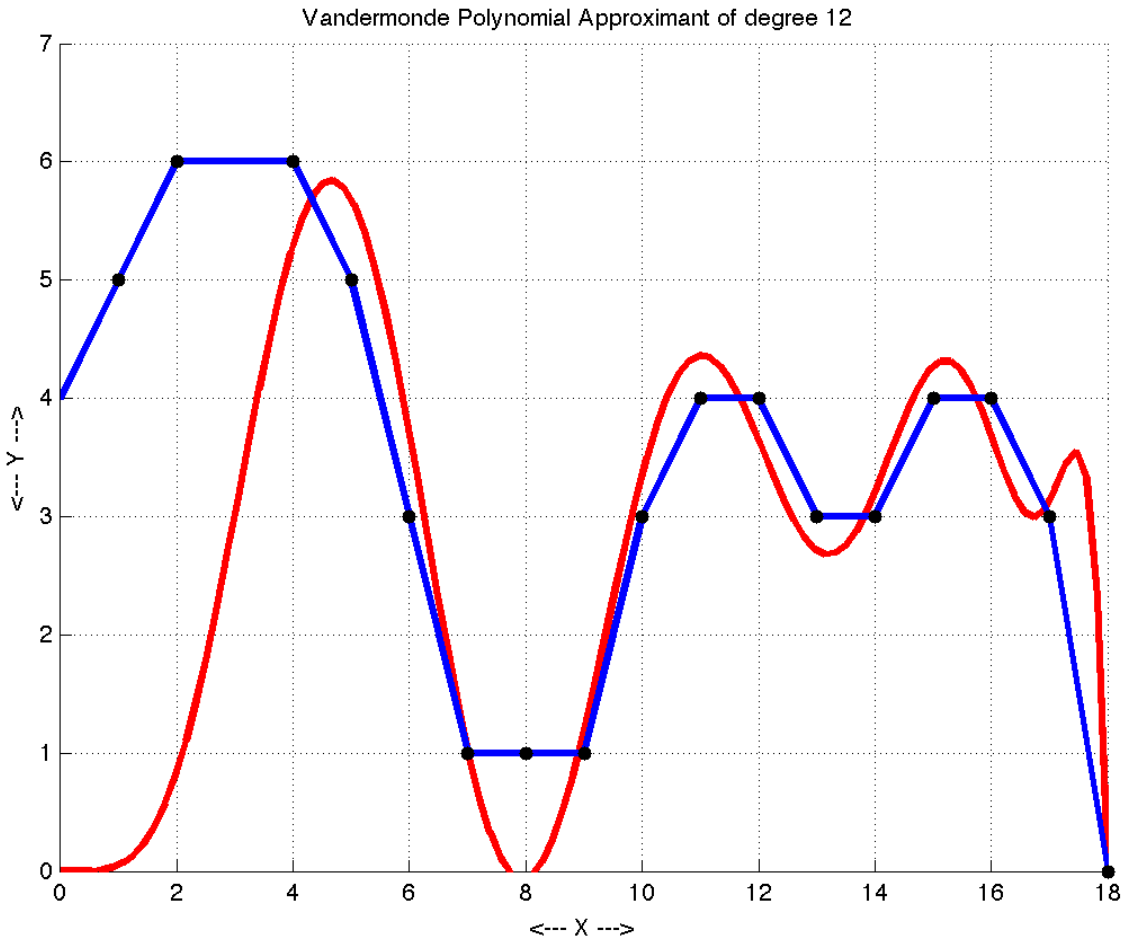
VANA: Vandermonde Approximant N = 11

Warning: Rank = 10



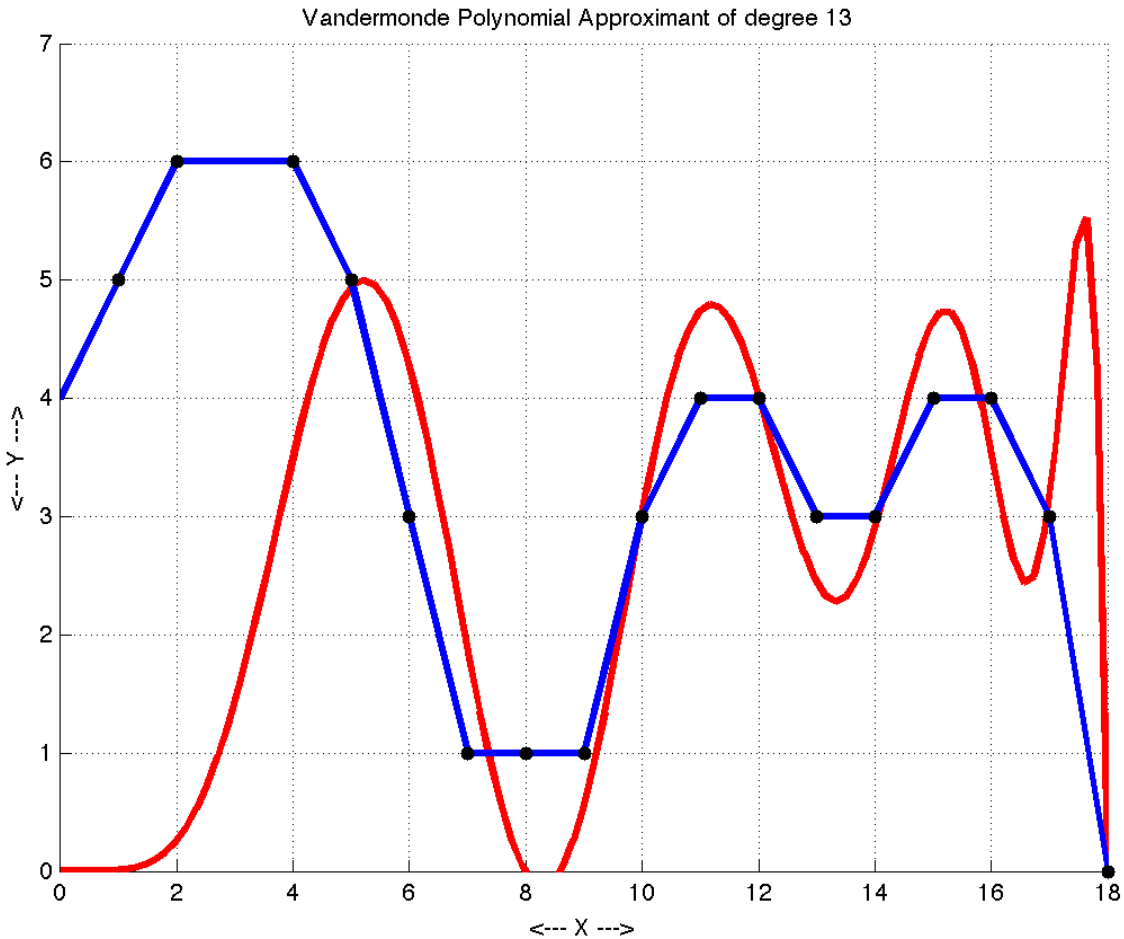
VANA: Vandermonde Approximant N = 12

Warning: Rank = 8



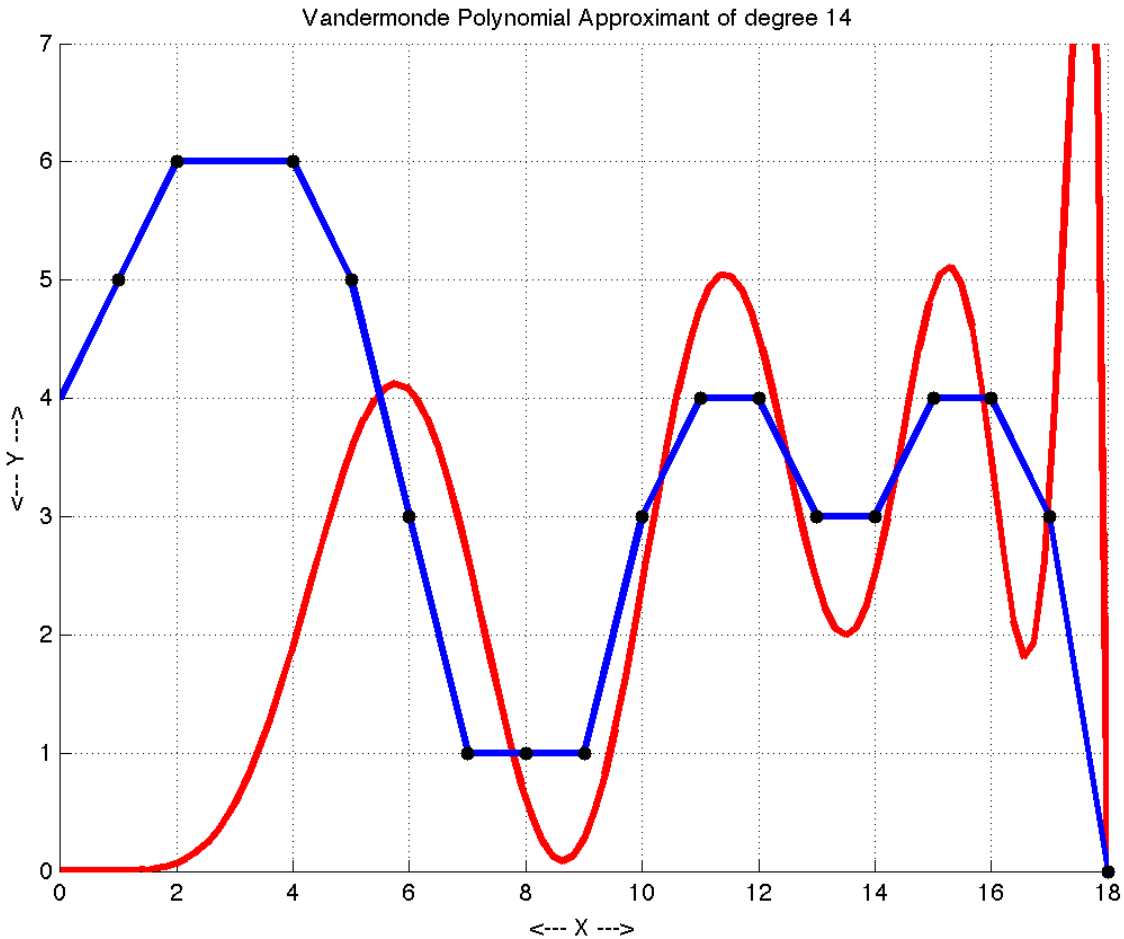
VANA: Vandermonde Approximant N = 13

Warning: Rank = 8



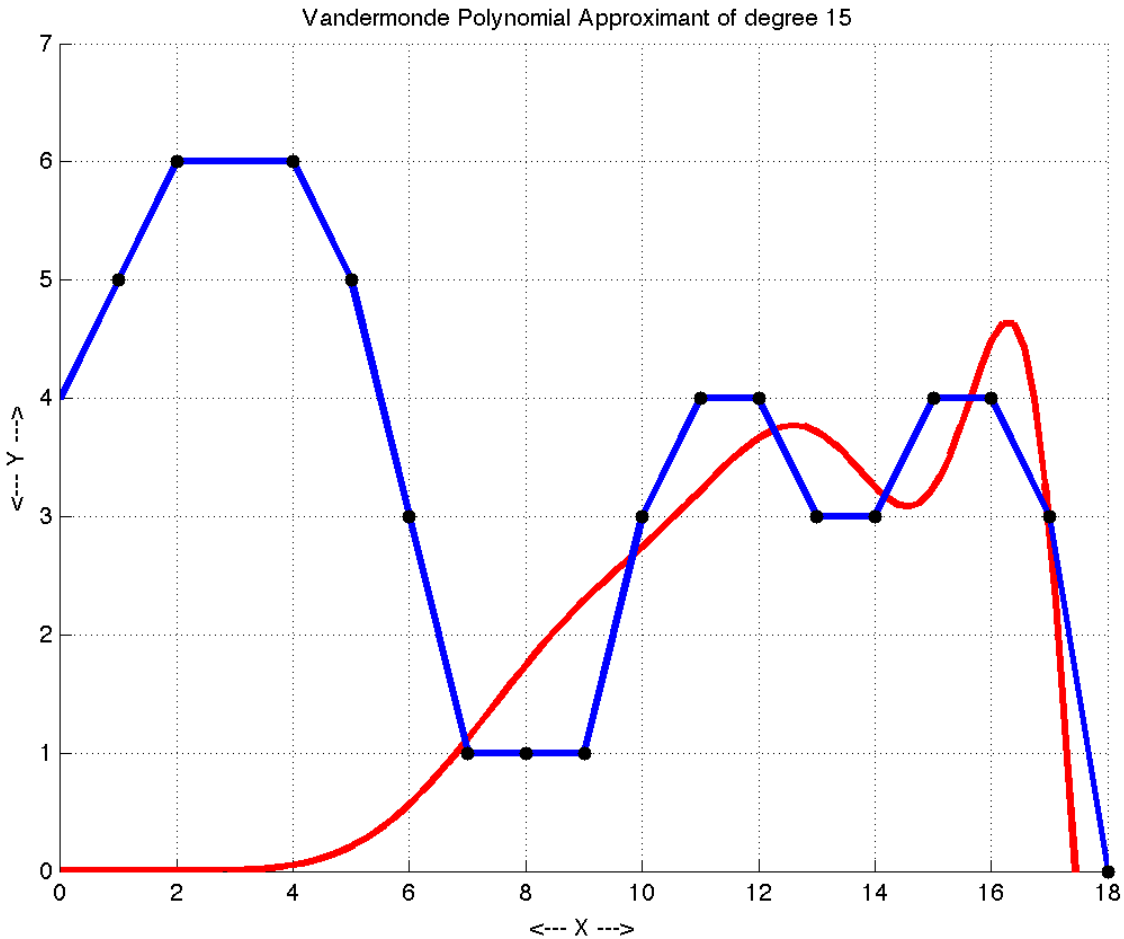
VANA: Vandermonde Approximant N = 14

Warning: Rank = 8



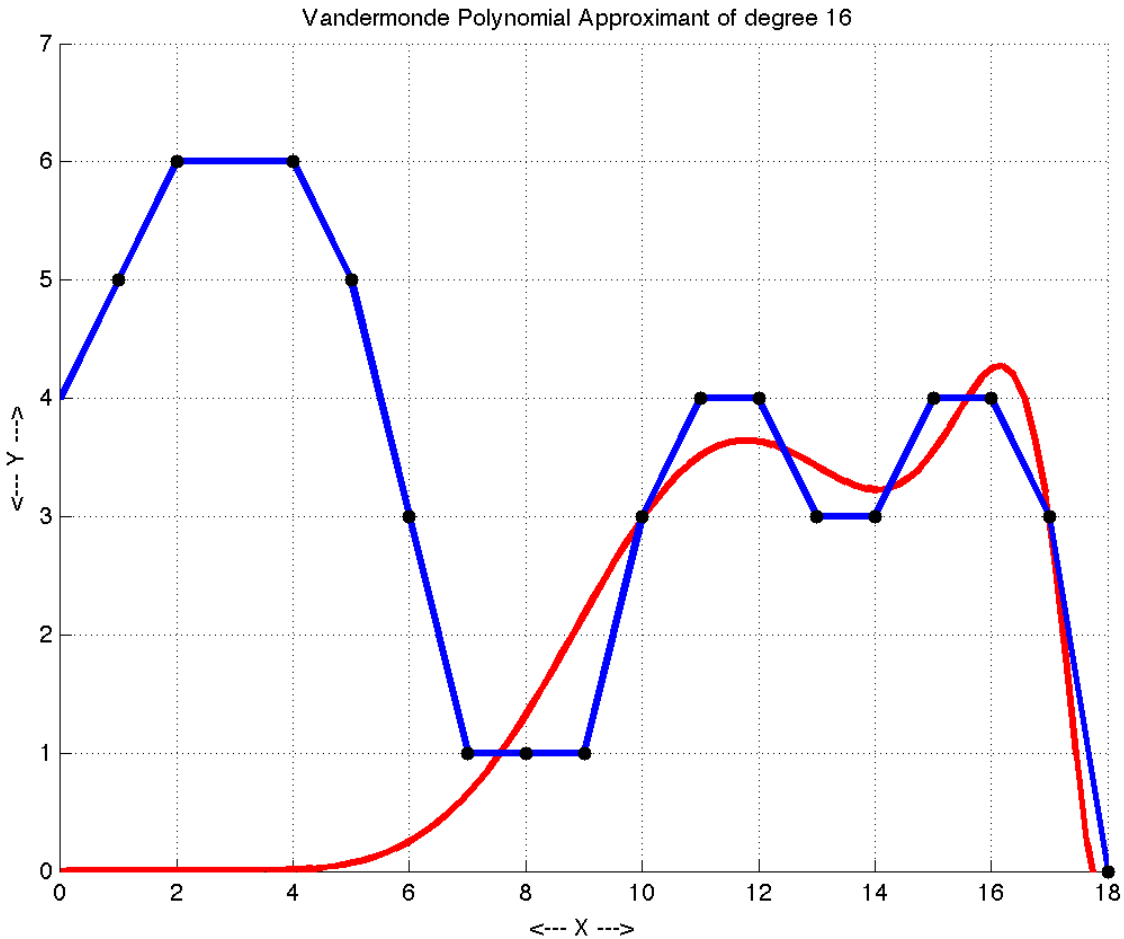
VANA: Vandermonde Approximant N = 15

Warning: Rank = 7



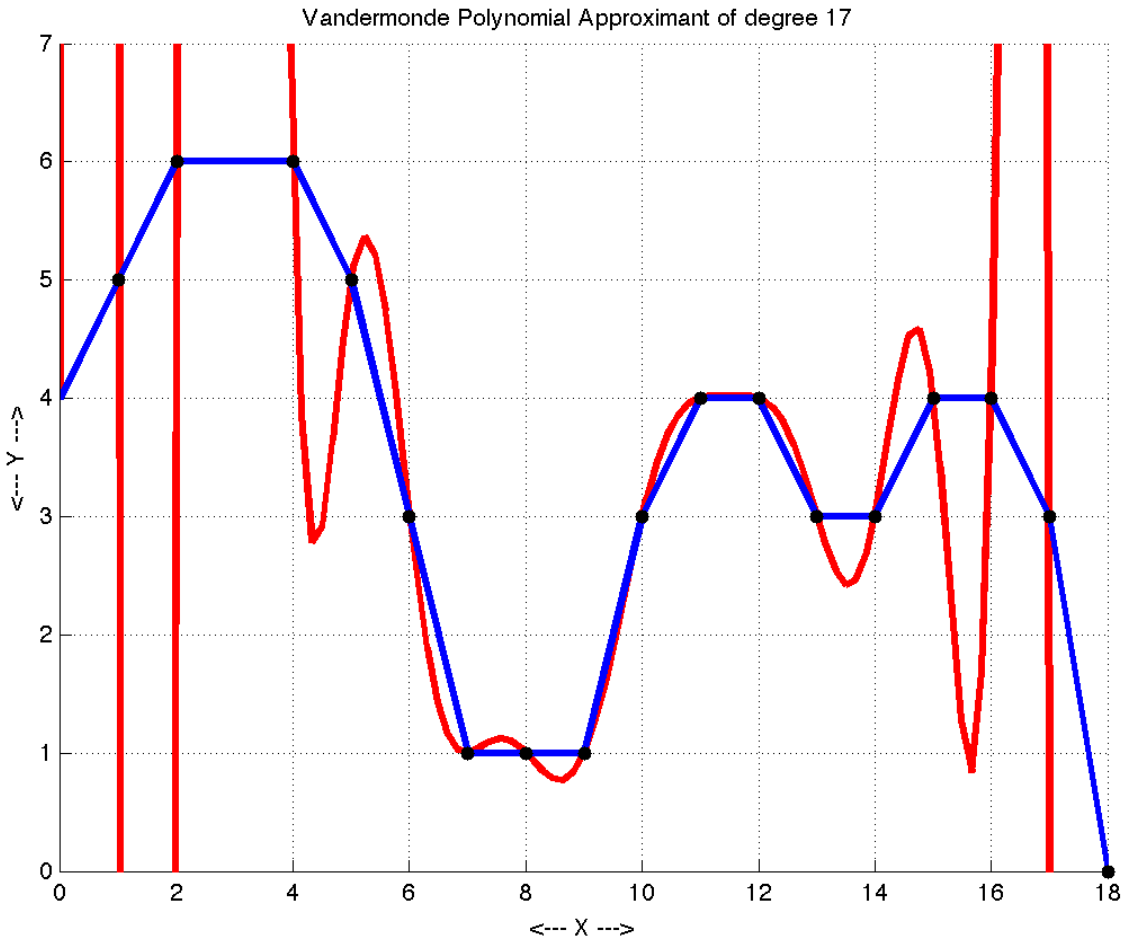
VANA: Vandermonde Approximant N = 16

Warning: Rank = 7



VANA: Vandermonde Approximant N = 17

Warning: Matrix condition is 10^{27}



VANA: Comments

Our approximation approach seemed to be going well, and it was clear that we could have stopped early and gotten an approximant that was close, not very oscillatory, and not badly scaled.

If our least squares matrix hadn't lost column rank, we could have had more confidence in this approach.

The problem afflicting the least squares matrix involves our choice of the monomials $1, x, x^2, ..., x^m$ as our basis functions for polynomials. Even when the interpolation nodes are well separated, it becomes very difficult to distinguish the monomial functions of high degree.

The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- **Lagrange Interpolation: Even Nodes**
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

LAGE: Lagrange Basis Functions (Even Spacing)

The Lagrange interpolation procedure has the same goal as we saw with the Vandermonde interpolation approach, namely, *find the degree $n-1$ polynomial interpolant for n data values.* The interpolating space $\mathcal{G}$ is the same, but the basis will differ.

Our basis will depend on the data locations $\{x_i\}$. We want our i-th basis function $\phi_i(x)$ to have the property that it is 1 at x_i and 0 at the other data locations. That way, there should be no problem distinguishing the basis functions!

$$\phi_i(x_j) = \delta_{i,j}$$

If we can make this happen, the interpolant is obvious:

$$g(x) = \sum_{i=1}^{n} y_i \, \phi_i(x)$$

LAGE: Lagrange Basis Functions (Even Spacing)

It's possible to write a formula for for $\phi_i(x)$:

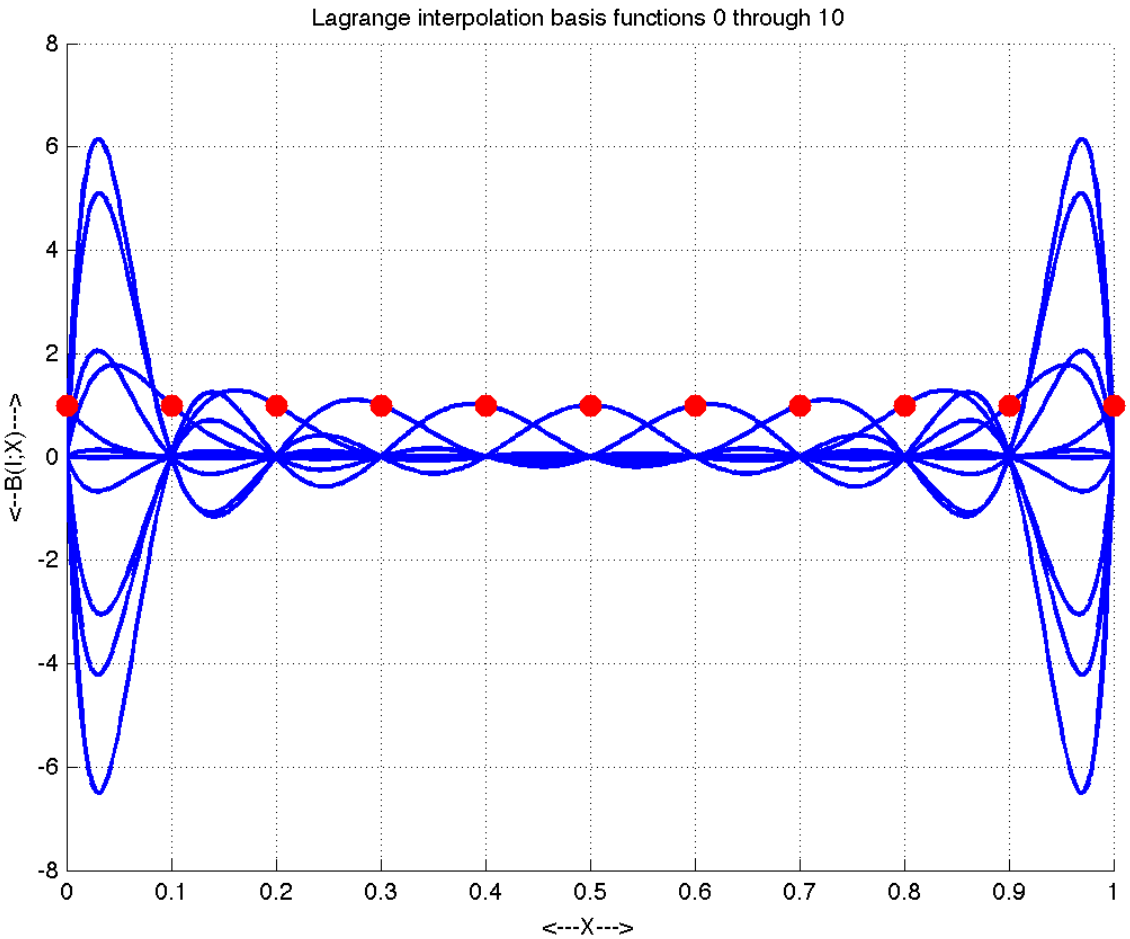
$$\phi_i(x) = \frac{\prod_{j=1, j\neq i}^{n}(x - x_j)}{\prod_{j=1, j\neq i}^{n}(x_i - x_j)}$$

Obviously, it is necessary that the data locations be distinct.

It's **not** obvious whether the arrangement or spacing of the nodes has an effect on the results.

So far, we have not addressed this issue. Let's assume the choice of the number of nodes and their locations are up to us. And let us make the sensible choice of equally spaced nodes.

LAGE: Lagrange Basis Functions (Even Spacing)



LAGE: Lagrange Basis Functions (Even Spacing)

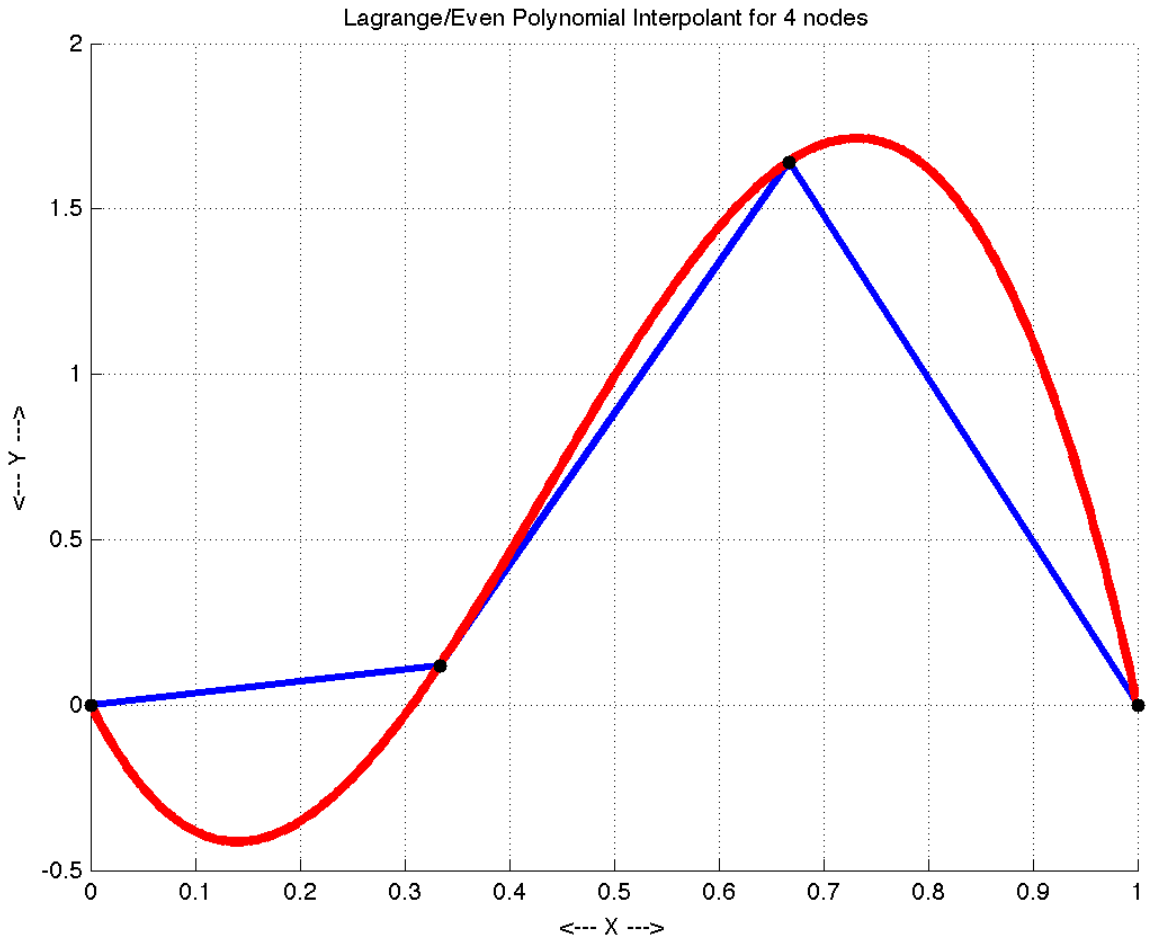
Again, the picture of the basis functions can be interpreted as a representation of the interpolation matrix entries.

We can see that at node x_i, basis function $\phi_i(x)$ is 1, and all others are zero. The interpolation matrix is the identity matrix.

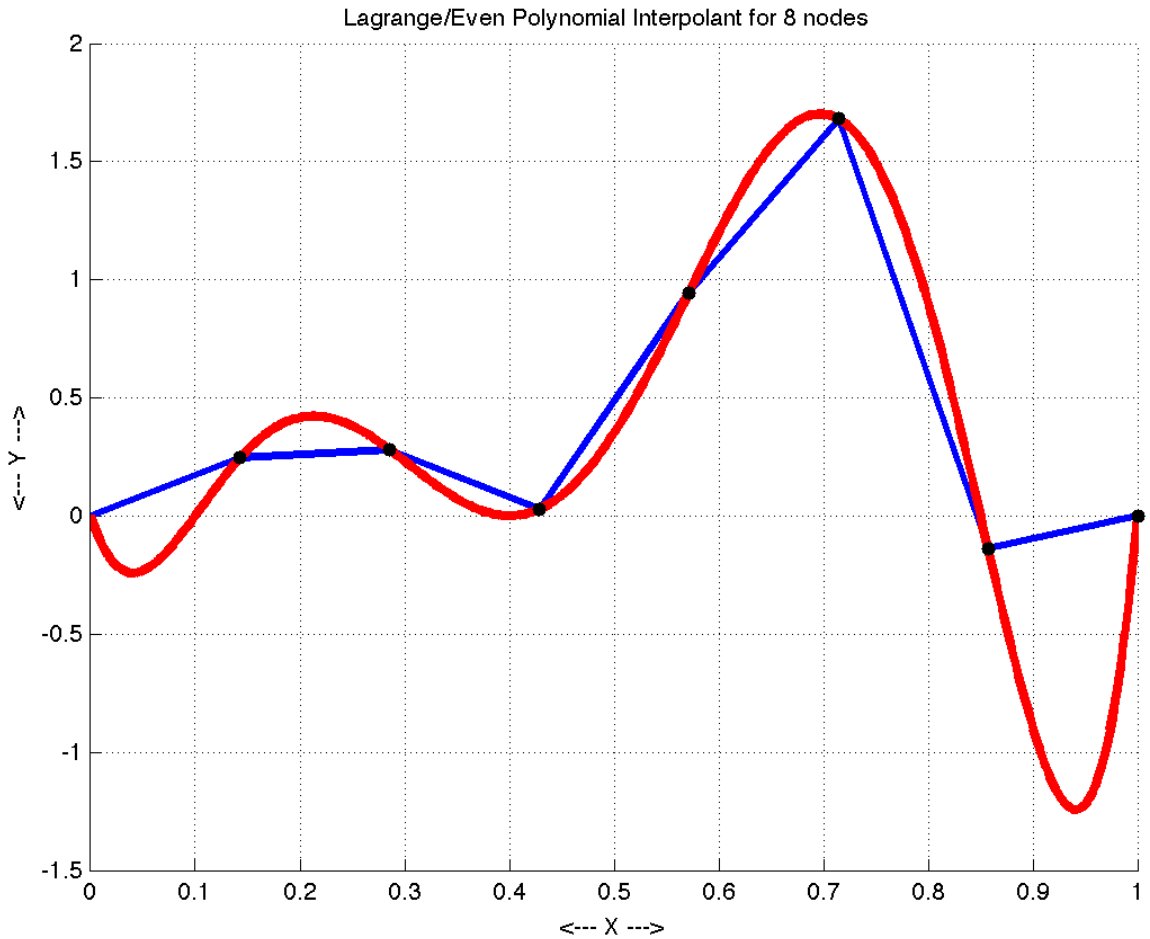
For our evenly spaced nodes, the plot also suggests that all the basis functions are "well behaved" in the interior, but tend to grow in size near the endpoints. This suggests that a small error in a basis function coefficient might be magnified at a non-interpolation point near the endpoints.

While this seems a minor problem at first, as we go to higher degree polynomials, the imbalance can explode.

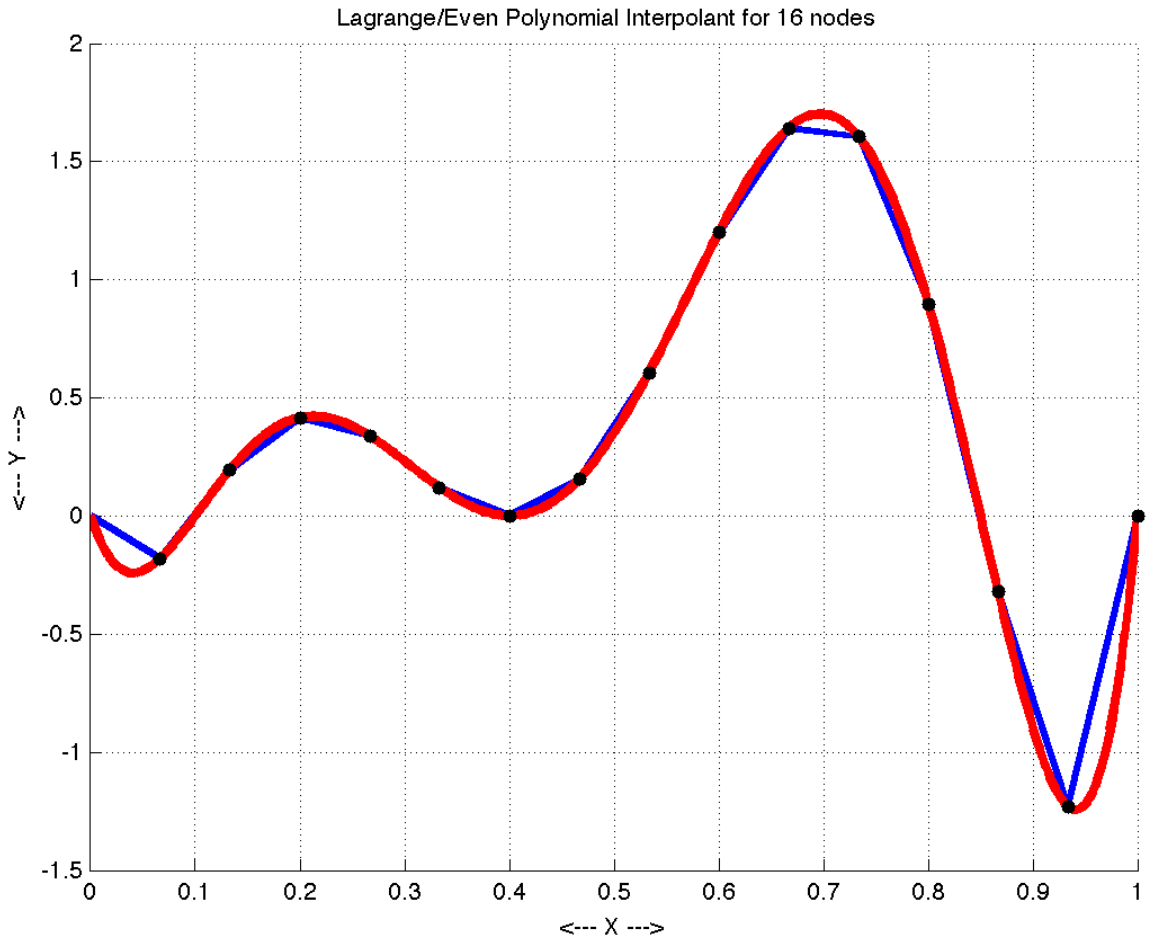
LAGE: Lagrange/Even Interpolant N+1 = 4



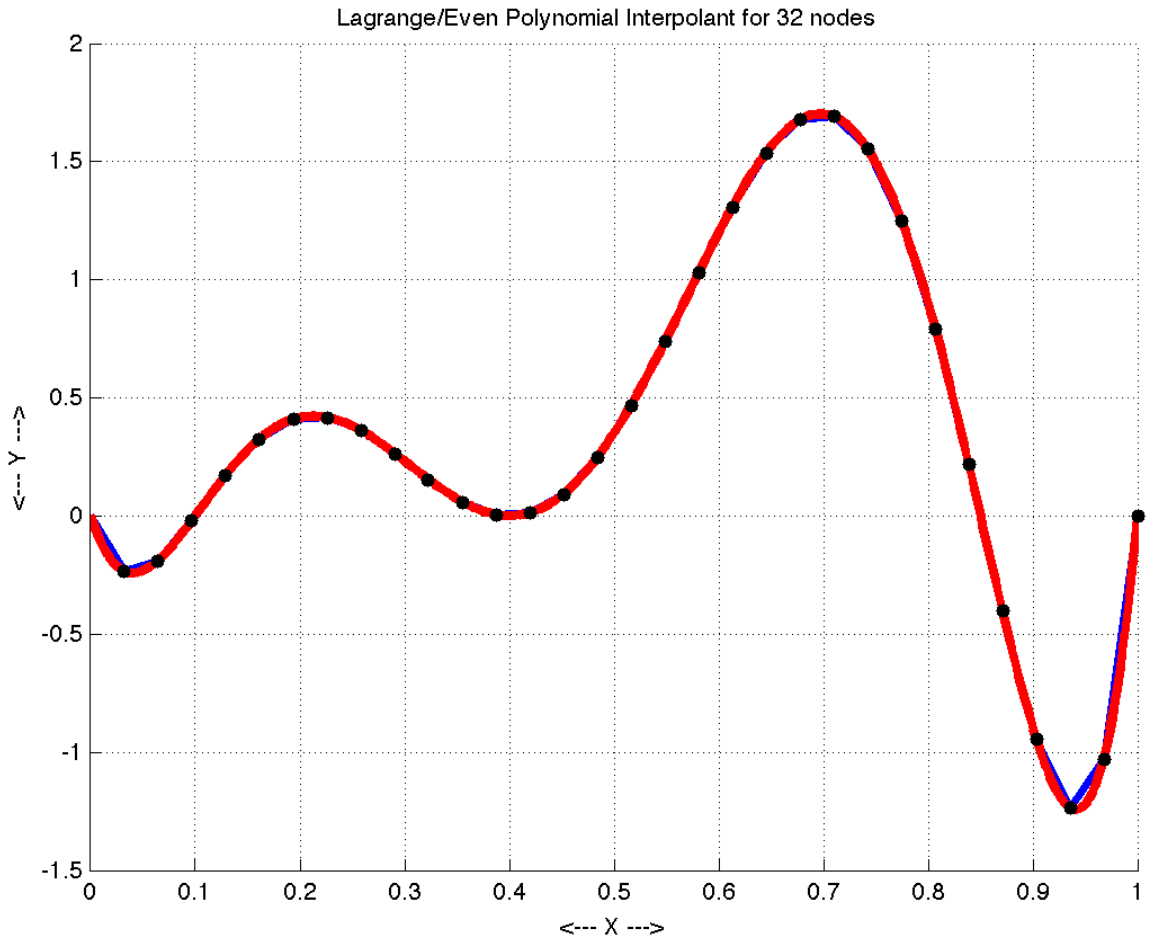
LAGE: Lagrange/Even Interpolant N+1 = 8



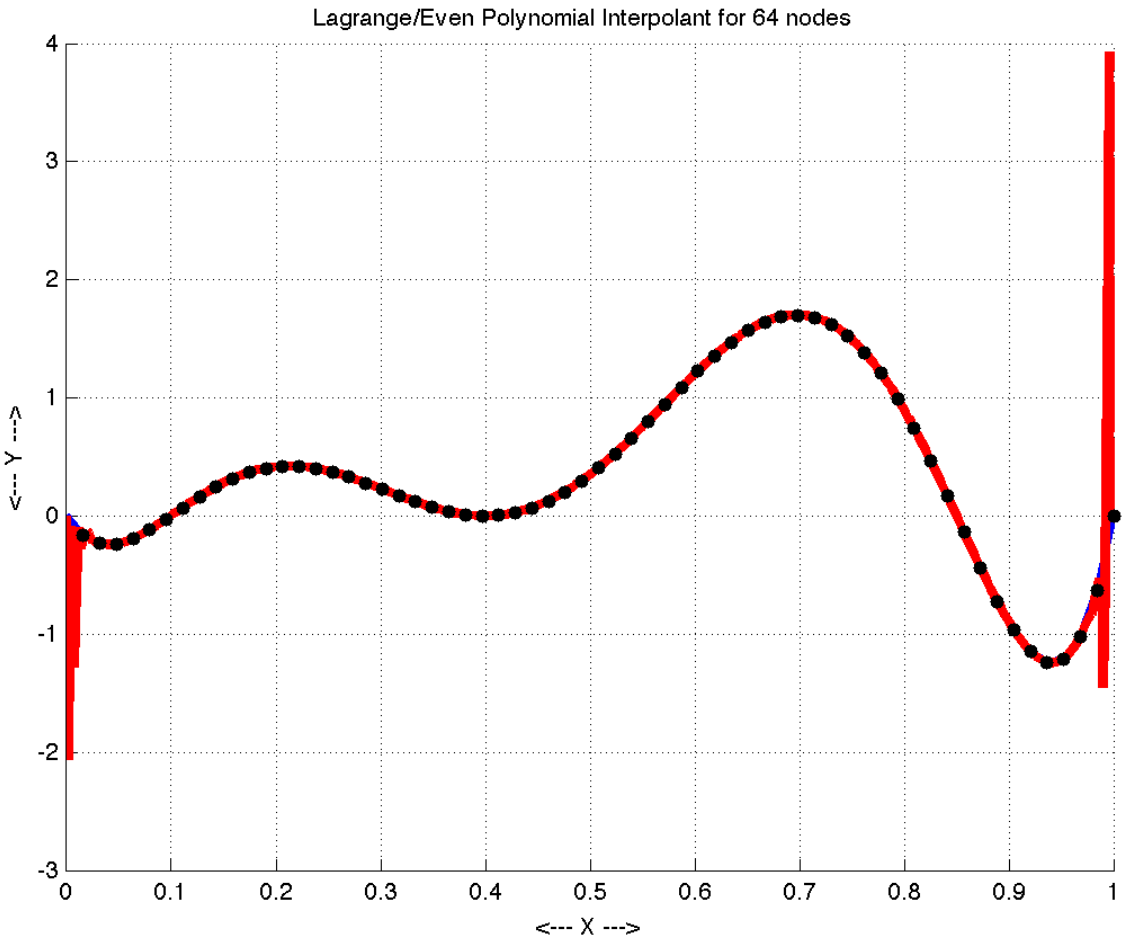
LAGE: Lagrange/Even Interpolant N+1 = 16



LAGE: Lagrange/Even Interpolant N+1 = 32



LAGE: Lagrange/Even Interpolant N+1 = 64



The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- **Lagrange Interpolation: Chebyshev Nodes**
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

LAGC: Lagrange Basis (Chebyshev Spacing)

If we are able to choose our interpolation points, then a common alternative to even spacing is the Chebyshev points, which are simply the cosines of evenly spaced angles.

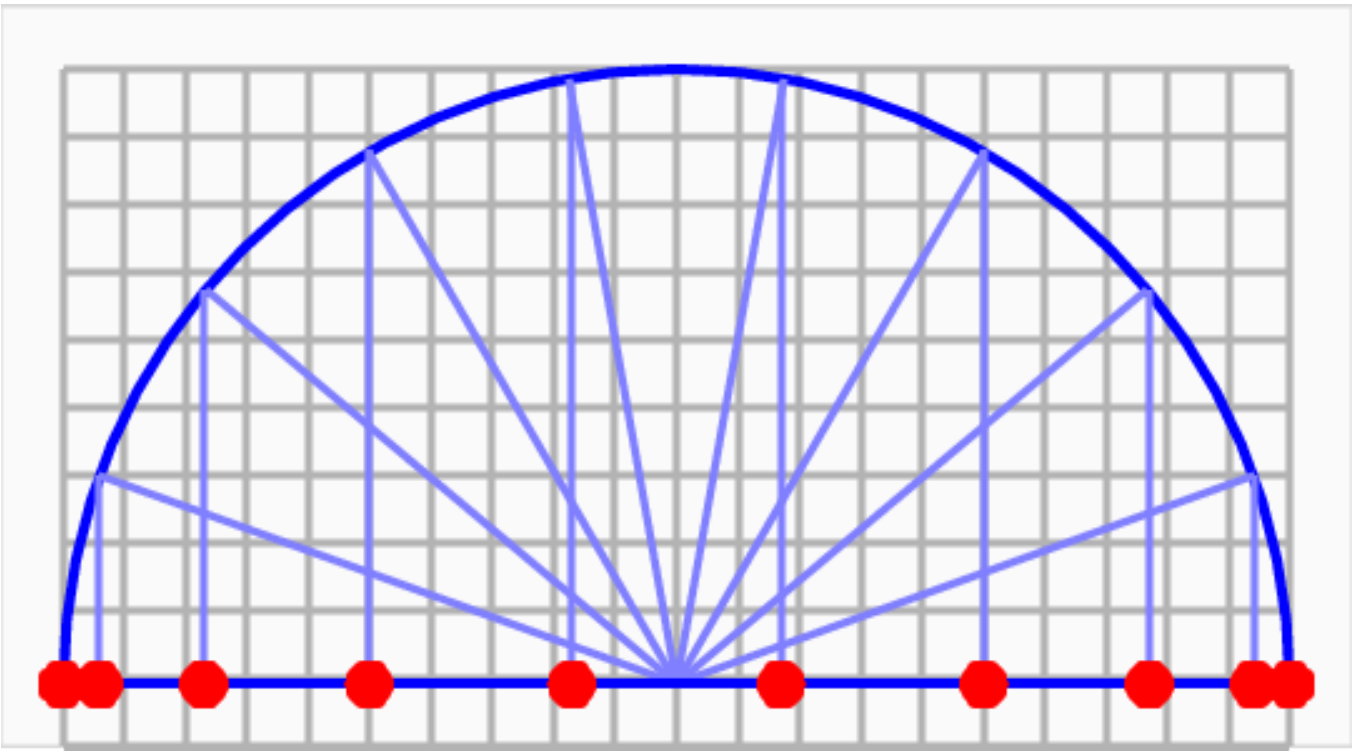
Since we can use any set of distinct nodes for the Lagrange procedure, it is still true that at node x_i, basis function $\phi_i(x)$ is 1, and all others are zero.

The new feature is that the basis functions are strongly localized (their mass is concentrated around their interpolation node) and throughout the rest of the region, their values are small. This means that we can avoid the possibility that a small error in interpolation results in a large error at some far away point.

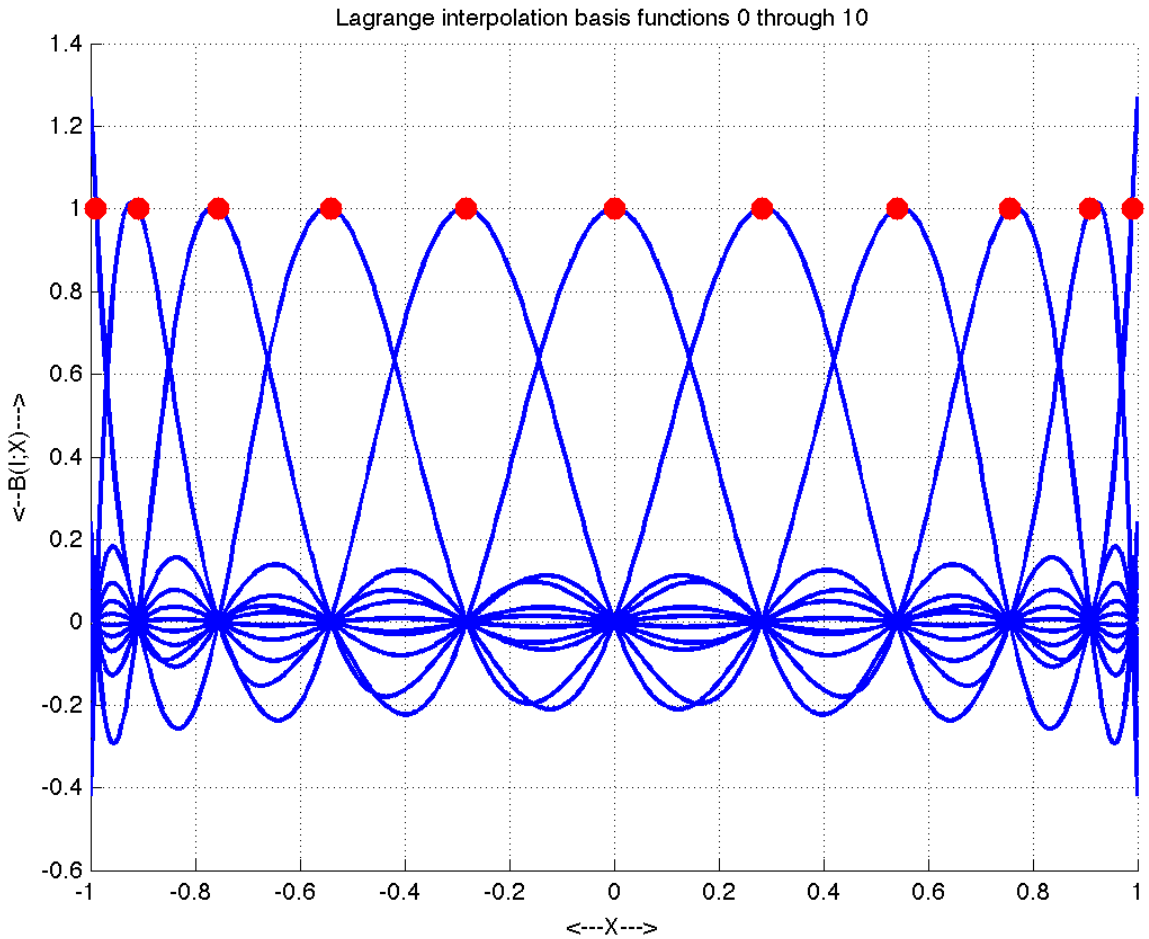
LAGC: Lagrange Basis (Chebyshev Spacing)

The derivation of the Chebyshev (Type 2) nodes:

$$x_i = \cos(\frac{(i-1)\pi}{n-1}), i = 1, ..., n$$



LAGC: Lagrange Basis (Chebyshev Spacing)



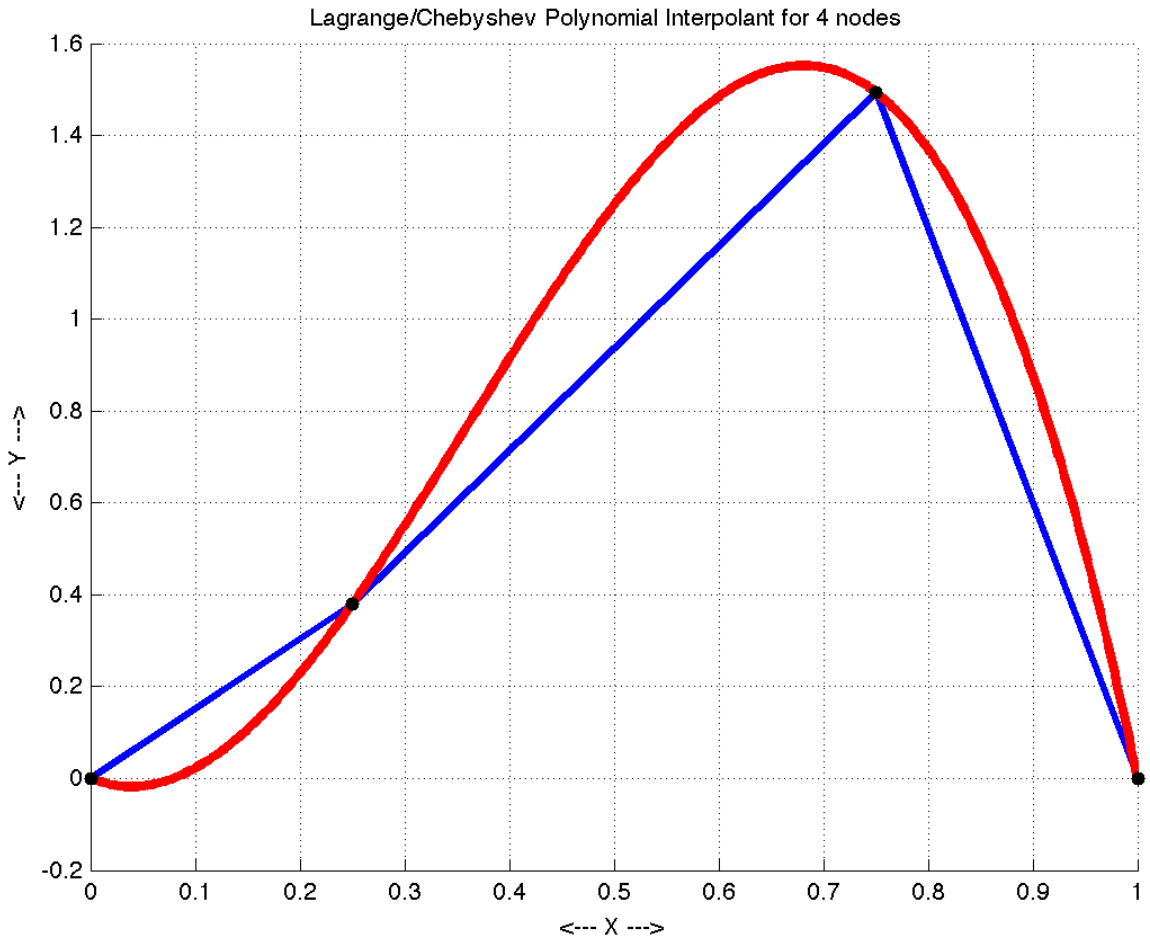
LAGC: Lagrange Basis (Chebyshev Spacing)

If we repeat the interpolation attempt for the same function as before, we do not get the error blowing up around the end intervals this time.

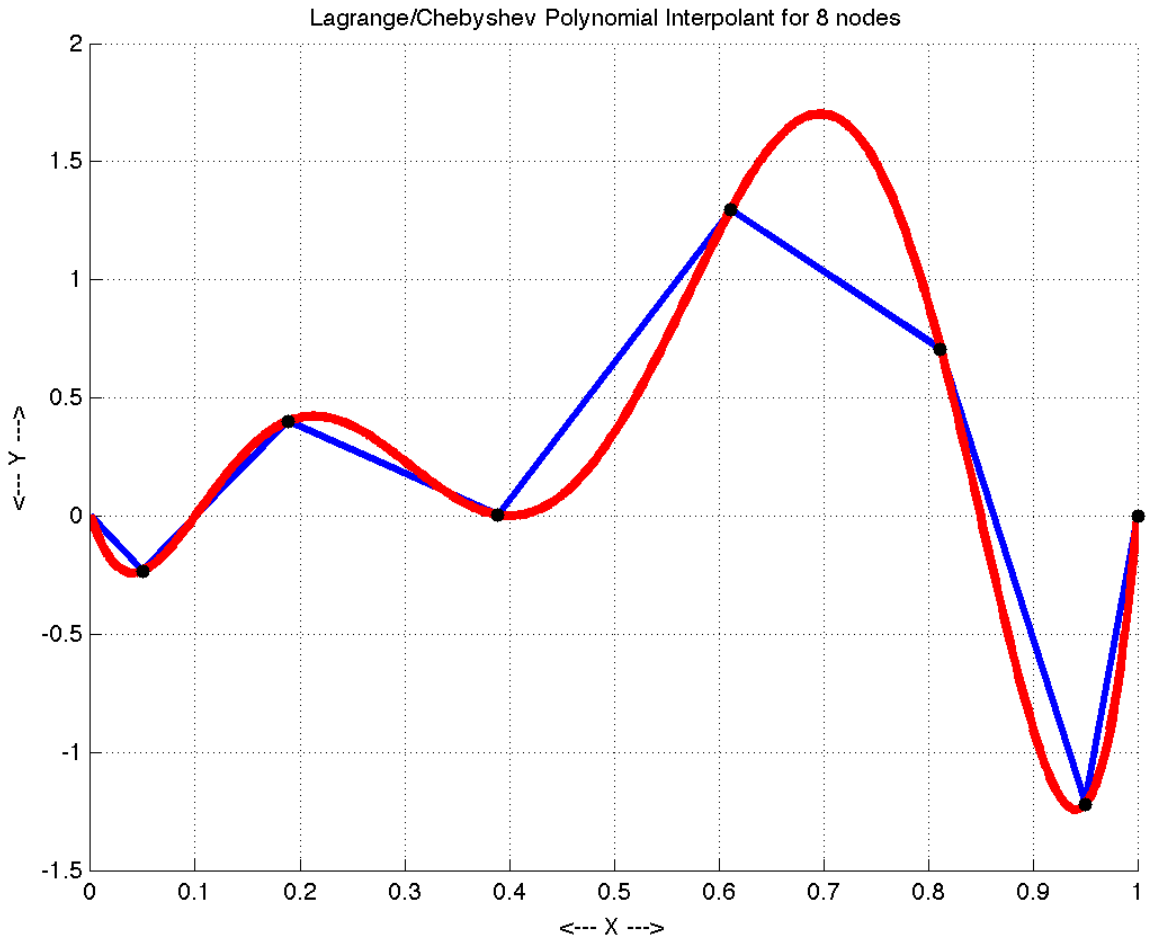
That is because the choice of Chebyshev nodes allows us to avoid the imbalance that would otherwise occur.

This suggest that high order Lagrange interpolation on an **arbitrary** set of nodes is likely to be inaccurate and unstable.

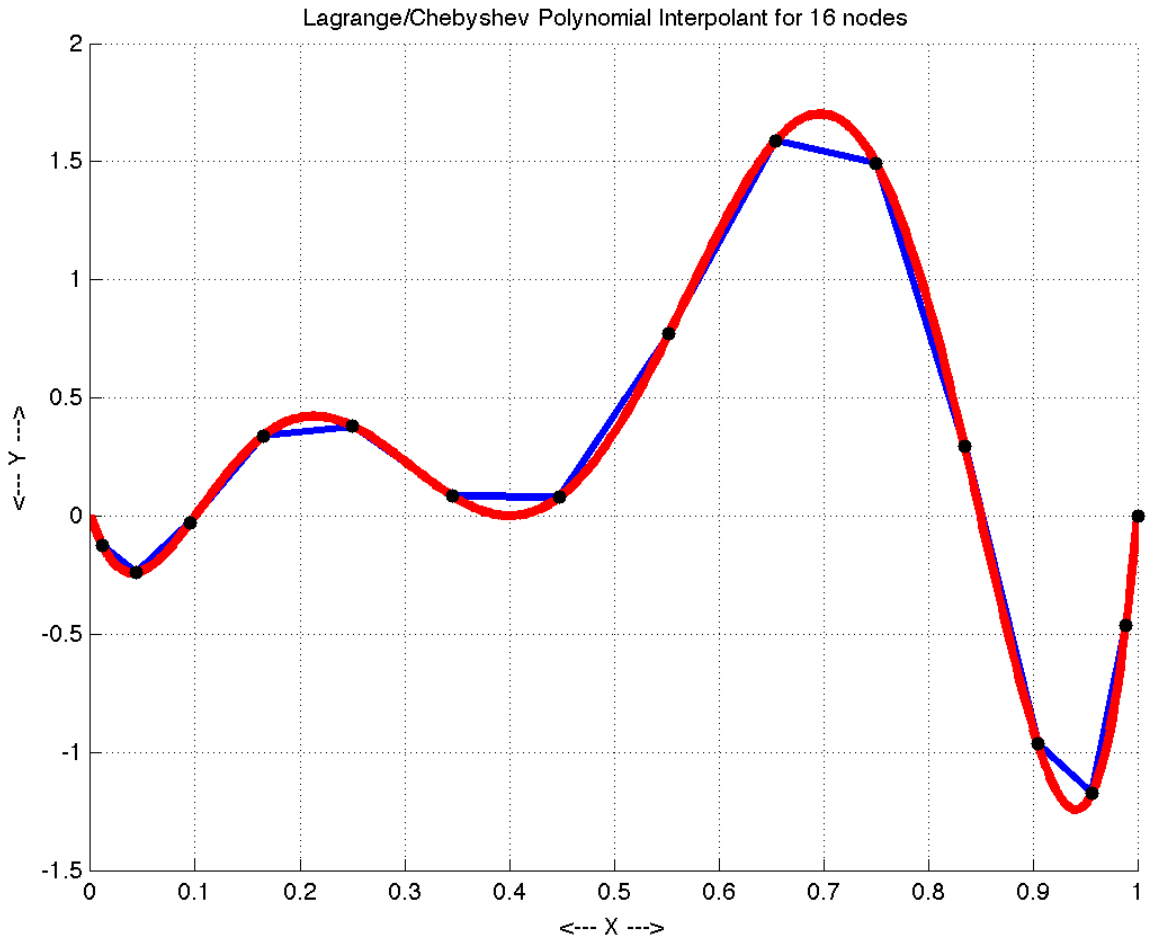
LAGC: Lagrange/Chebyshev Interpolant N+1 = 4



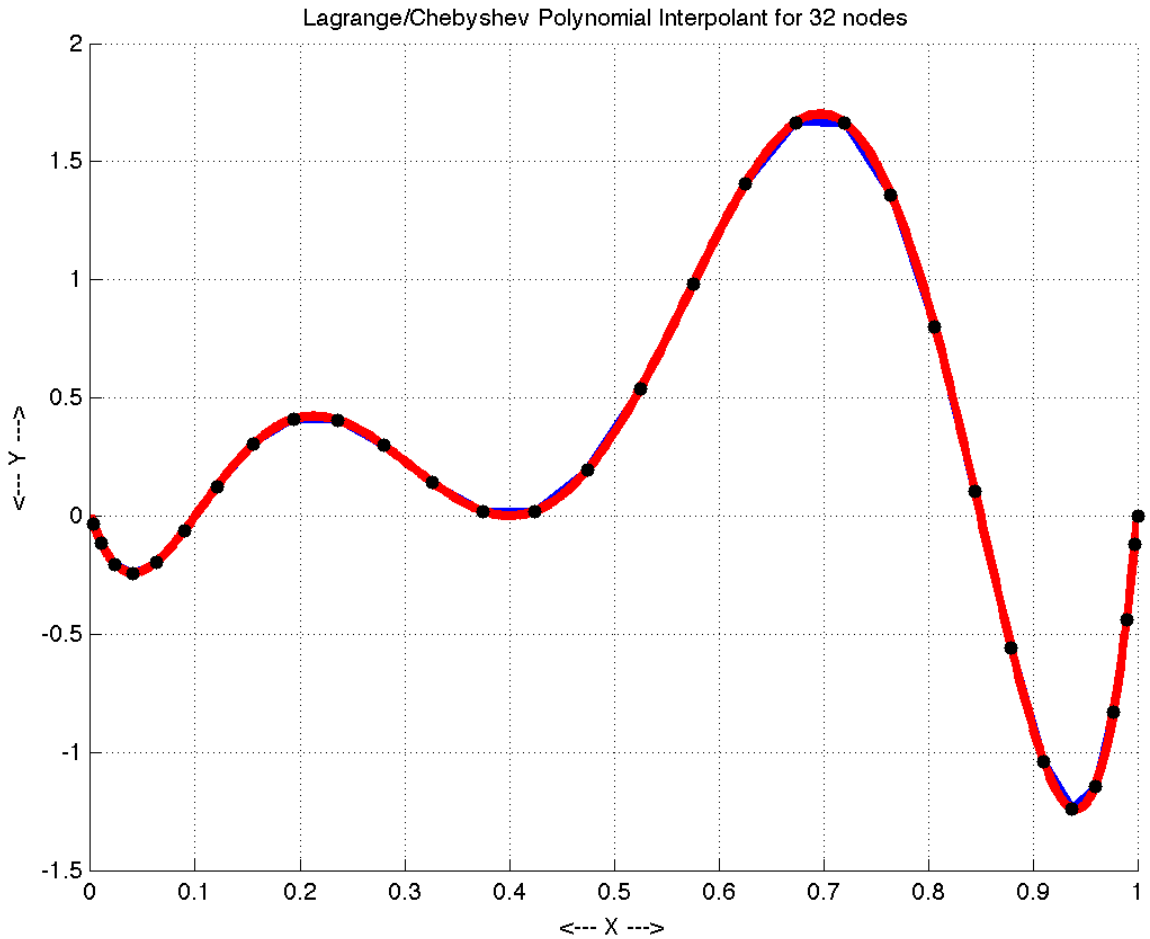
LAGC: Lagrange/Chebyshev Interpolant N+1 = 8



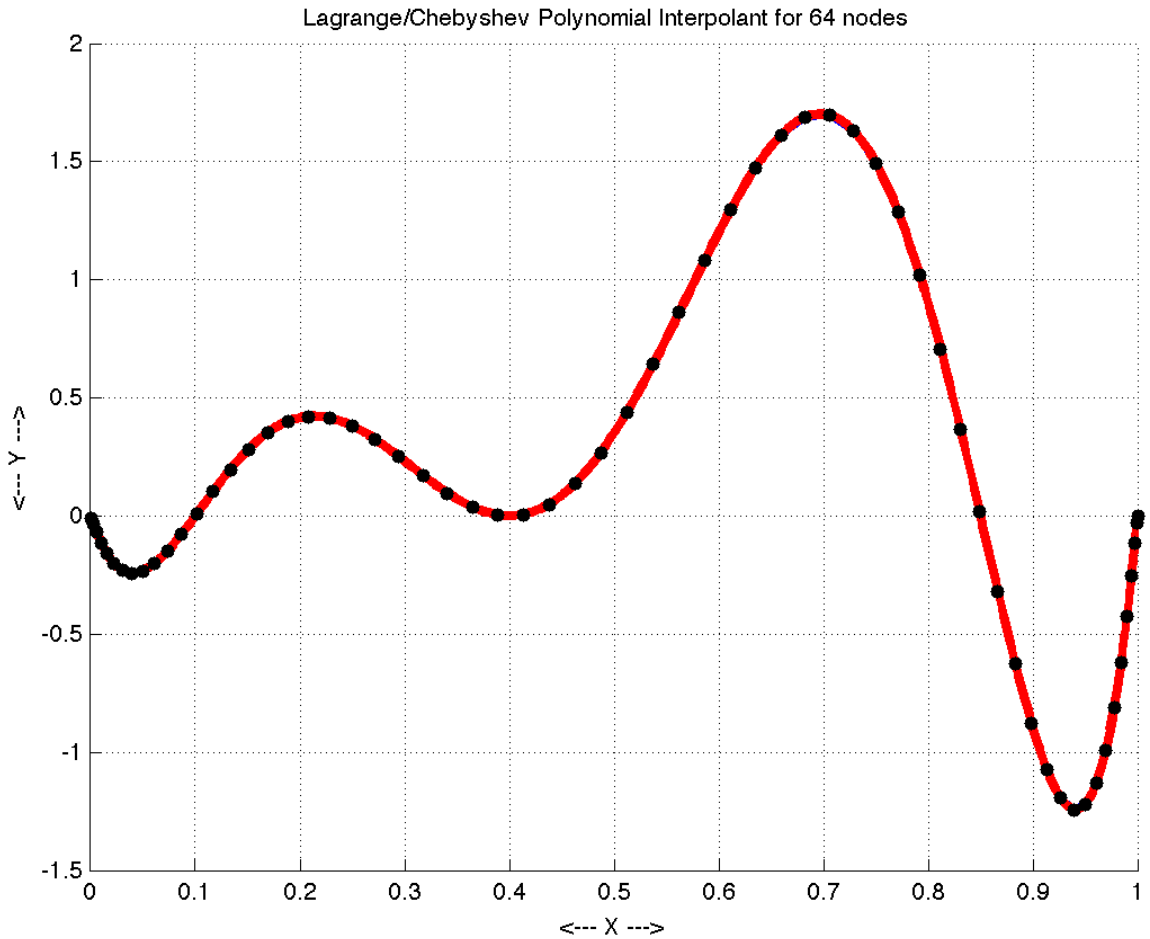
LAGC: Lagrange/Chebyshev Interpolant N+1 = 16



LAGC: Lagrange/Chebyshev Interpolant N+1 = 32



LAGC: Lagrange/Chebyshev Interpolant N+1 = 64



The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- **Lagrange Approximation**
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- Radial Basis Functions

LAGA: Lagrange Approximation

We have seen that the Lagrange formulation of the interpolating polynomial can handle cases with 64 data points, but we must expect numerical instability as the number of data points increases.

What if we are given 1000 data points to interpolate?

Assuming the interpolant will become impossible to evaluate accurately, we'd consider constructing an approximant. We have done this for the case where the polynomial is determined in the Vandermonde procedure. Can we do it for the Lagrange formulation instead?

LAGA: Lagrange Approximation

To construct a Lagrange approximant, we have to assume that there are a set of *nd data points* to approximate, and a set of *nc control points* that define the approximant.

Given 1000 data points in the interval [A,B], our approach would be to

- select Chebyshev control locations $\{xc_i\}$ in [A,B];
- represent the approximant as $g(x) = \sum_{i=1}^{nc} yc_i \, l_i(x)$;
- write down the *nd* approximation equations in *nc* variables as $A * yc = yd$
- solve the system in a least squares sense.

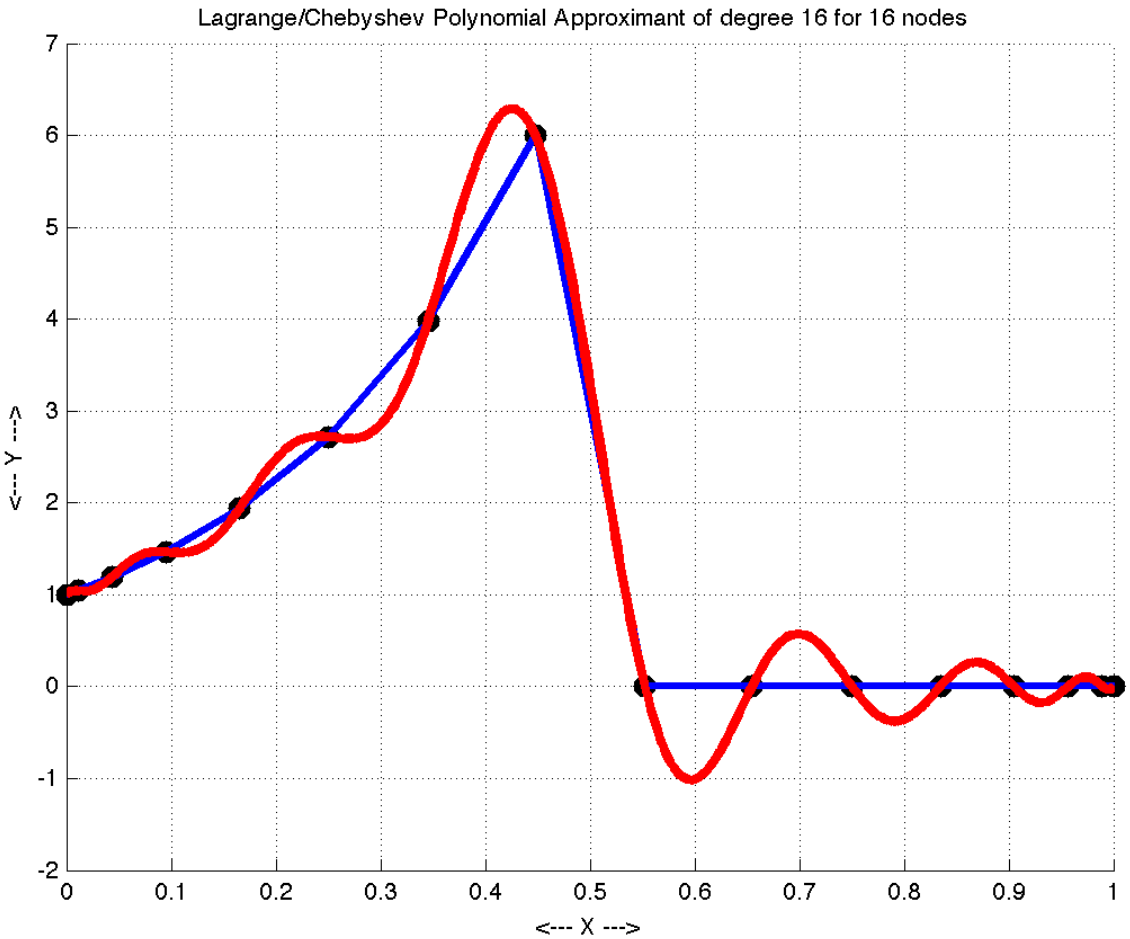
LAGA: Lagrange Approximation

In the interpolation problem, the matrix A was the identity matrix, because we were evaluating the function at the control points, where exactly one basis function is nonzero.

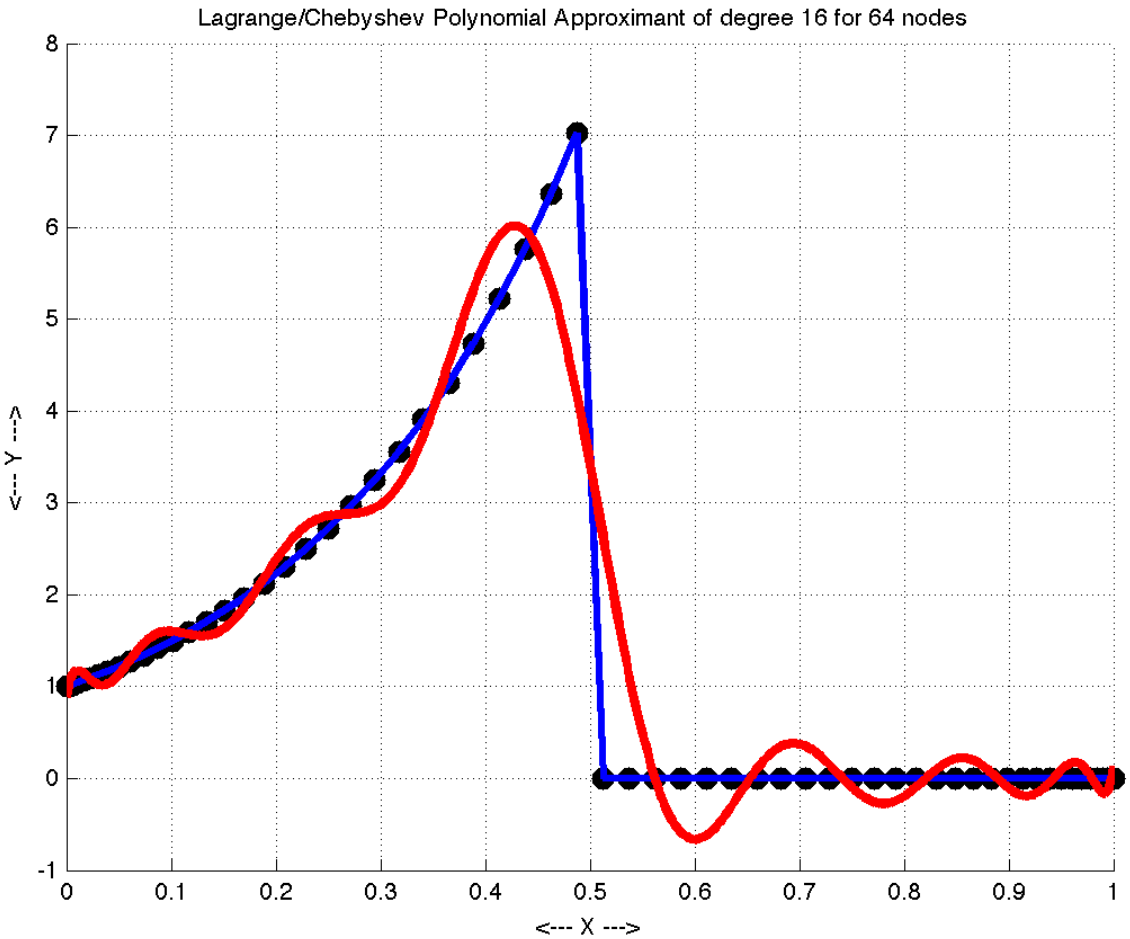
So by going to an approximation form, we incur the cost of setting up and solving a dense rectangular $(nd \times nc)$ linear system.

However, we hope to gain a numerically stable low-order approximant that has been determined by a high number of data values.

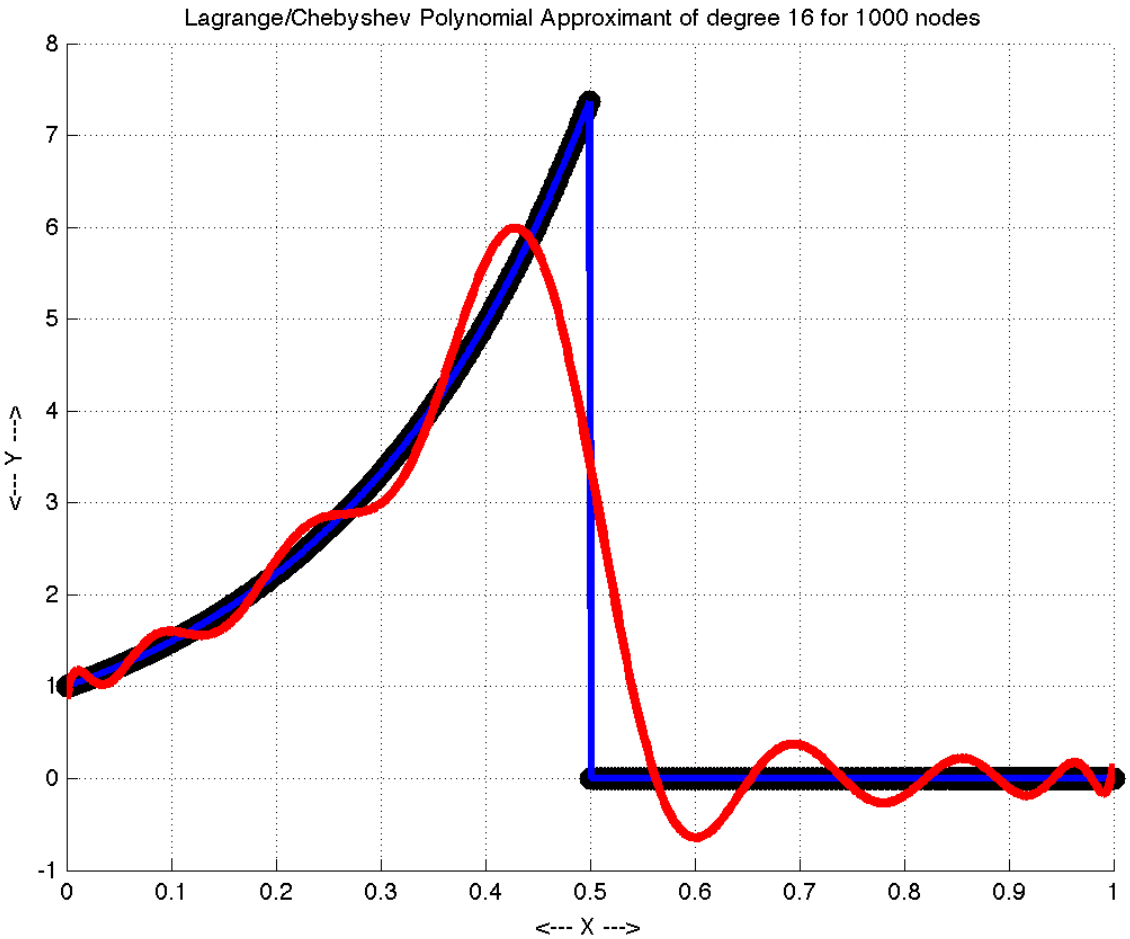
LAGA: Lagrange/Chebyshev, Degree 16, ND=16



LAGA: Lagrange/Chebyshev, Degree 16, ND=64



LAGA: Lagrange/Chebyshev, Degree 16, ND=1000



The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- **Barycentric Lagrange Interpolation**
- Shepard Interpolation
- Radial Basis Functions

BARY: The Barycentric Lagrange Formula

We know that using the Lagrange approach for the interpolation polynomial gives us a better basis than Vandermonde does.

We know that choosing points that are biased towards the interval endpoints gives us a more even set of weights, and hence a more stable calculation.

When the amount of data is small, we are comfortable using the standard Lagrange interpolant, but we are concerned about stability and accuracy if we try to interpolate 1000 data points with a polynomial of degree 999.

The *barycentric Lagrange interpolation formula* provides an efficient and stable scheme for evaluating such high-degree interpolants.

BARY: The Barycentric Lagrange Formula

If we let $\ell(x)$ be defined by

$$\ell(x) = \prod_{j=1}^{n} (x - x_j)$$

and write w_i, the barycentric weight for the j-th data value:

$$w_i = \frac{1}{\prod_{j \neq i} (x_i - x_j)}$$

the Lagrange interpolant is:

$$g(x) = \ell(x) \sum_{i=1}^{n} \frac{w_i}{x - x_i} y_i$$

BARY: The Barycentric Lagrange Formula

However, this formula can be rearranged so that the high degree products disappear, replaced by barycentric coefficients that sum to 1:

$$g(x) = \frac{\sum_{i=1}^{n} \frac{w_i}{x-x_i} f_i}{\sum_{i=1}^{n} \frac{w_i}{x-x_i}}$$

This calculation is more stable; moreover, for a given set of nodes, the barycentric coefficients do not depend on the data f_i, and hence can be computed once and used to interpolate as many sets of functions as desired.

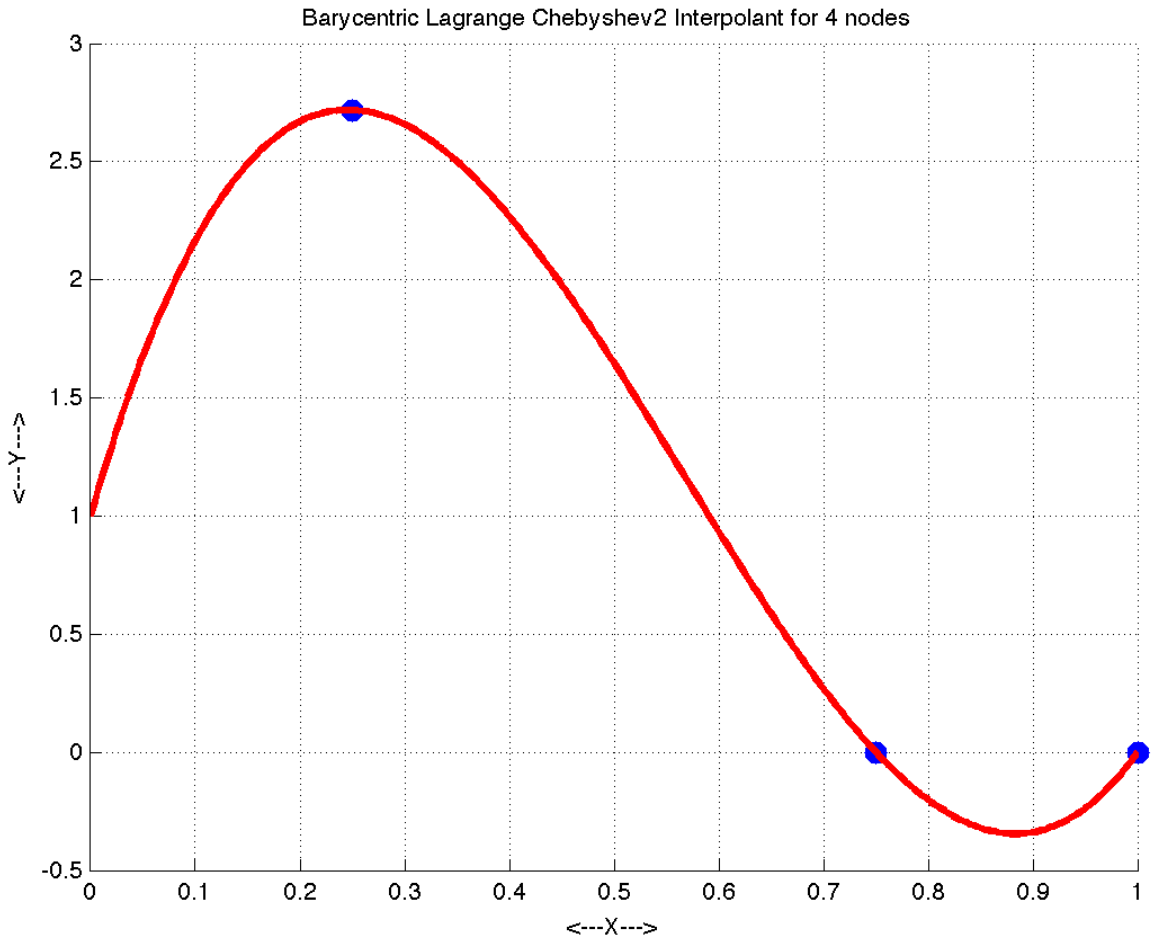
If the nodes are chosen to be Chebyshev points, the values of the w_i can also be computed directly from a formula.

BARY: Matlab Code for Chebyshev Type 2 Nodes

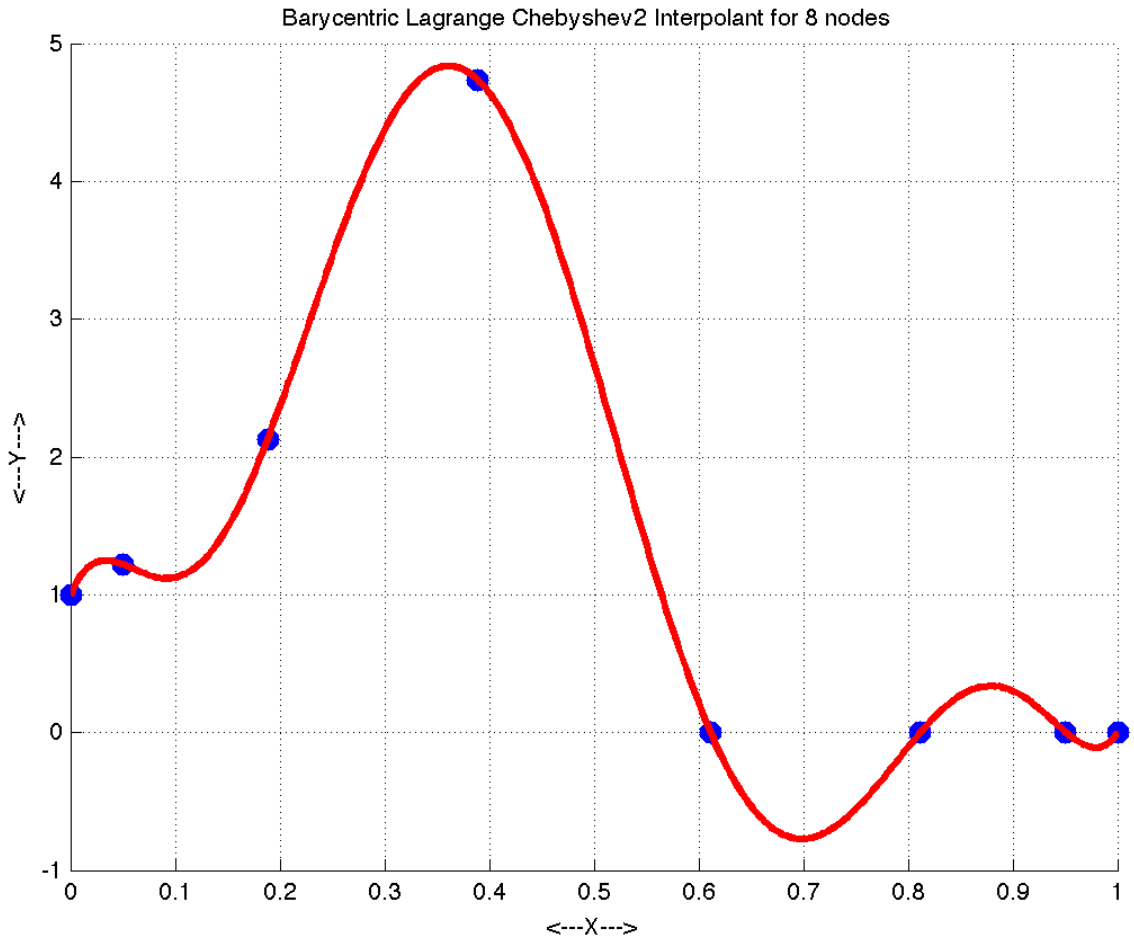
```
function yi = lagcheby2_interp_1d ( nd, xd, yd, ni, xi )

  wd = [ 1/2; ones(nd-2,1); 1/2 ] .* (-1) .^ ( (0:nd-1)' );

  numer = zeros ( ni, 1 );
  denom = zeros ( ni, 1 );
  exact = zeros ( ni, 1 );

  for j = 1 : nd
    t = wd(j) ./ ( xi - xd(j) );
    numer = numer + t * yd(j);
    denom = denom + t;
    exact( xi == xd(j) ) = j;      <-- Evaluation at a data node?
  end

  yi = numer ./ denom;

  j = find ( exact );              <-- Evaluated at data node?
  yi(j) = yd(exact(j));                Return data value there.

  return
end
```

From Berrut and Trefethen reference.

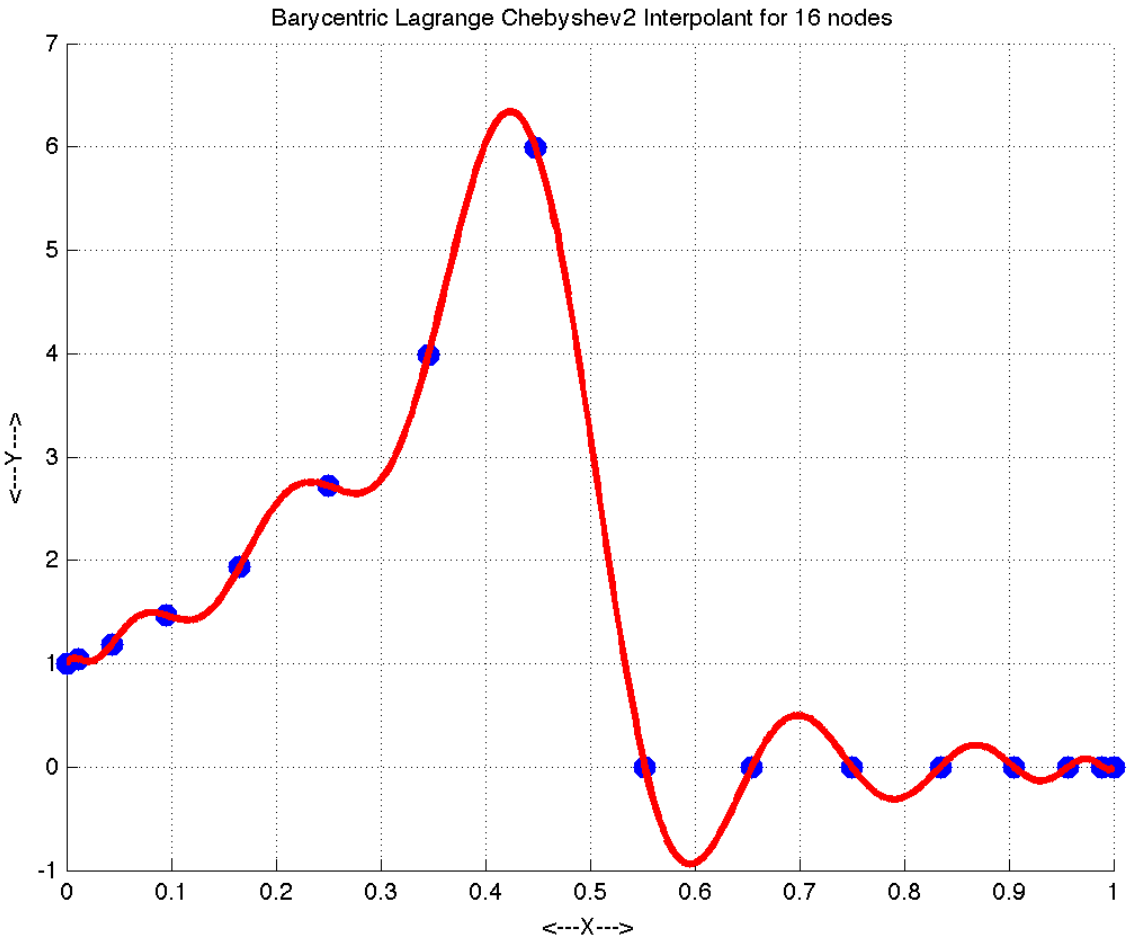
Bary: Barycentric, ND=4



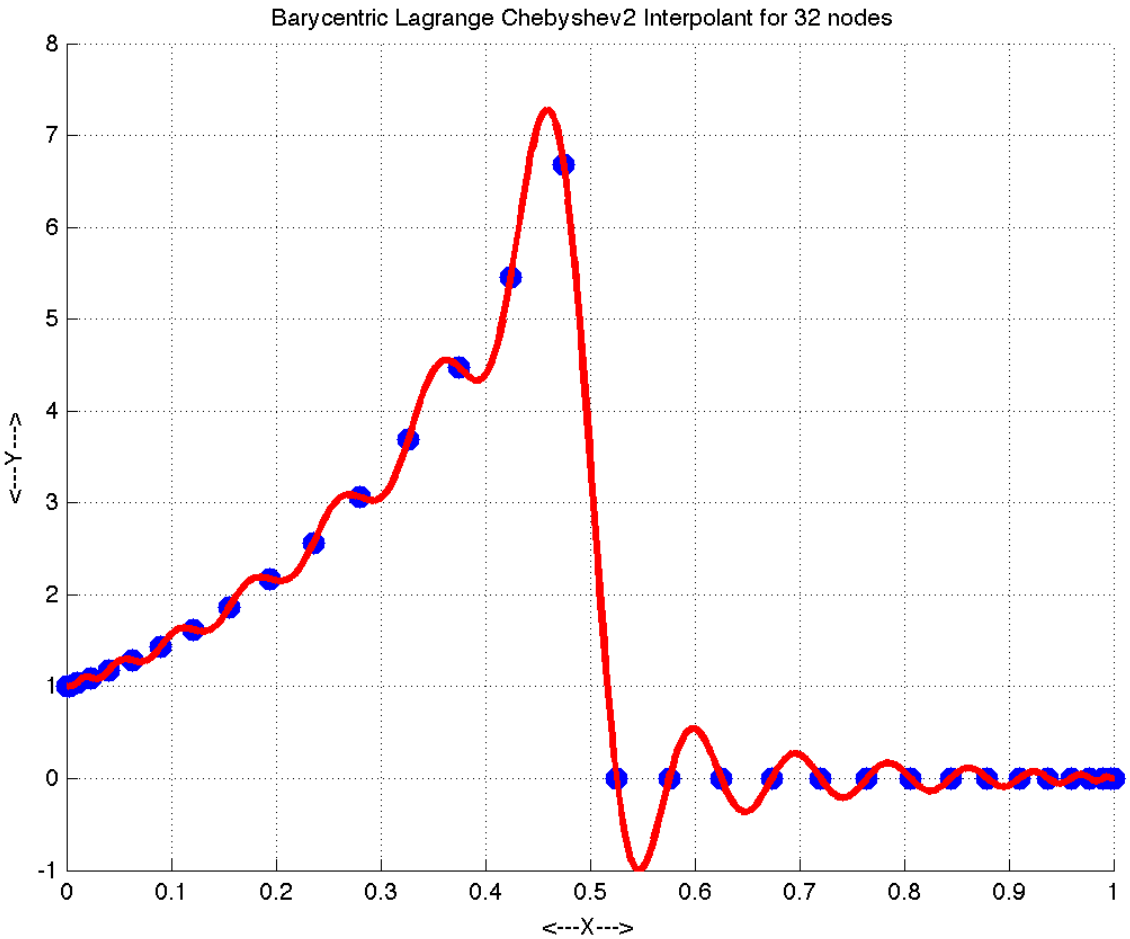
Bary: Barycentric, ND=8



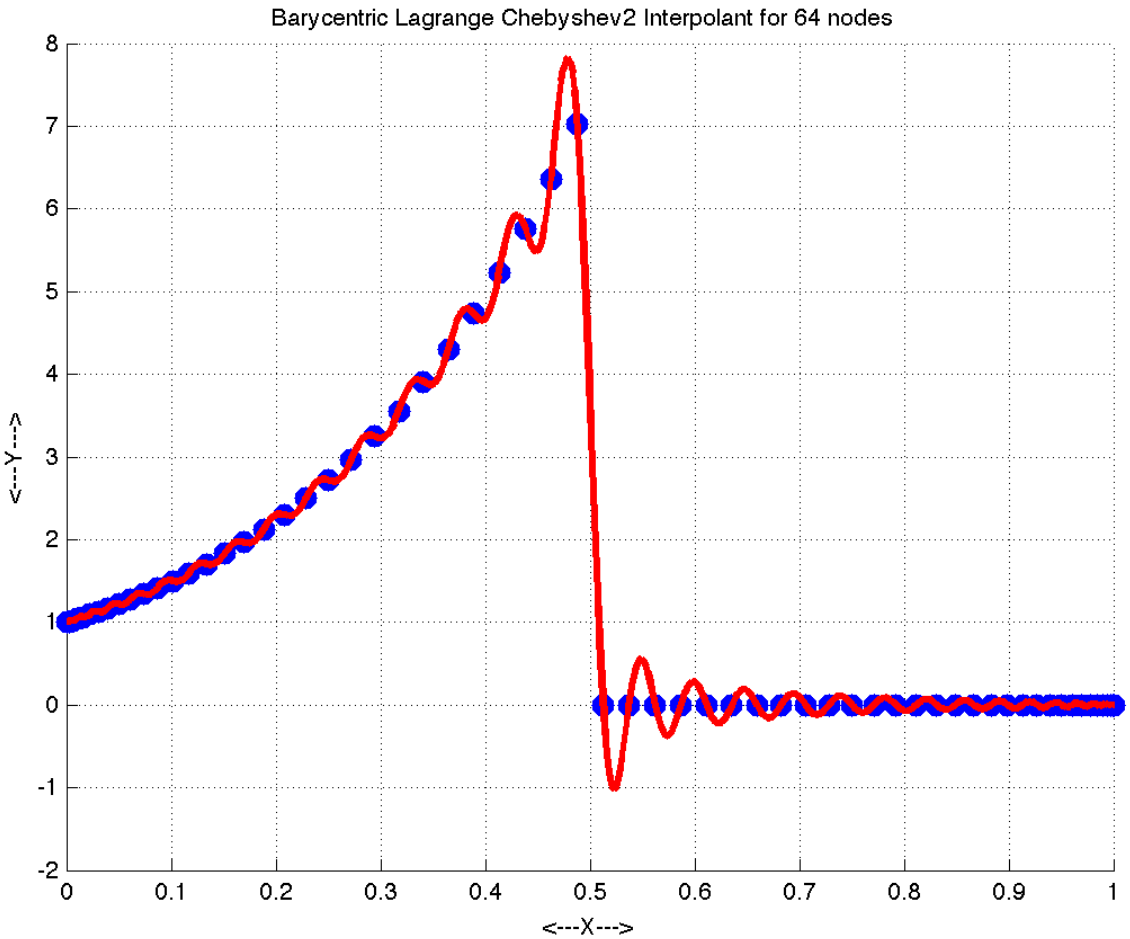
Bary: Barycentric, ND=16



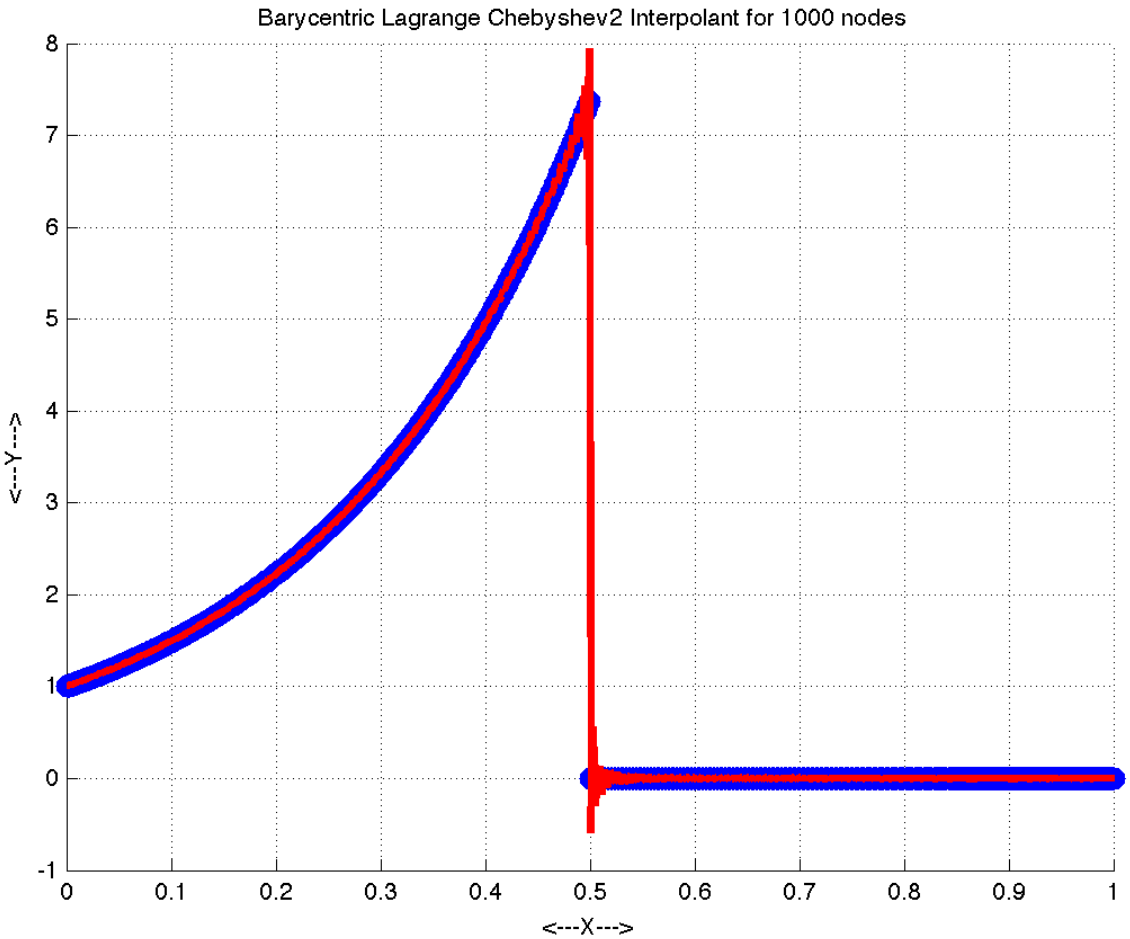
Bary: Barycentric, ND=32



Bary: Barycentric, ND=64



Bary: Barycentric, ND=1000



The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- **Shepard Interpolation**
- Radial Basis Functions

SHEP: Interpolation of Scattered Data

While the Lagrange interpolant is a powerful tool, in higher dimensions it is difficult to construct the basis functions unless the data can be sampled on a product grid.

Product grids in high dimensions become exponentially costly. It's worth looking at alternative approaches for interpolation.

One simple idea for interpolation of irregular or scattered data in an arbitrary dimension space is known as Shepard interpolation.

SHEP: Interpolation of Scattered Data

Shepard interpolation originated in the construction of smooth maps of scattered geographic data such as rainfall, altitude, population, soil composition.

The goal was to create a plausible model which agreed with the data, and varied continuously and differentiably in between, so that, at any point, the nearest measured data exerted the strongest influence on the model function.

(If we were willing to pay the cost of triangulating the data locations, we could construct a piecewise linear interpolant, but in higher dimensions, triangulation is out of the question.)

SHEP: The Shepard Formula

At an arbitrary sample point x, we determine the 'weight' or influence of the data point x_i by the formula

$$w_i(x) = \frac{1}{||x - x_i||^p}$$

We normalize the weights:

$$\hat{w}_i(x) = \frac{w_i(x)}{\sum_{j=1}^{n} w_j(x)}$$

and represent the interpolant by

$$g(x) = \sum_{j=1}^{n} \hat{w}_i(x) * y_i$$

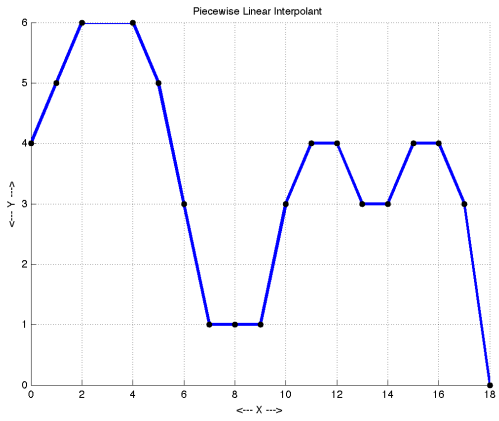
SHEP: The P Parameter

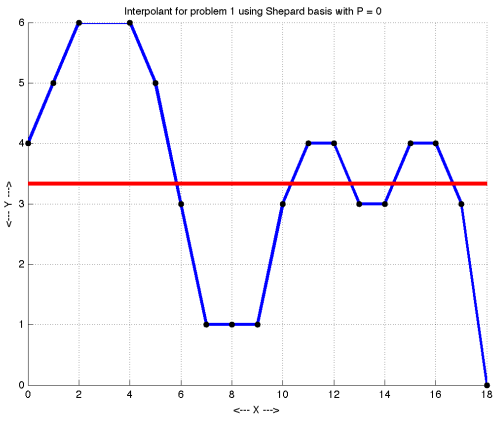
Here $||x - x_i||$ is any suitable distance measure, and p is a power which is often taken as 2. The value of p influences the relative importance of local and distant data.

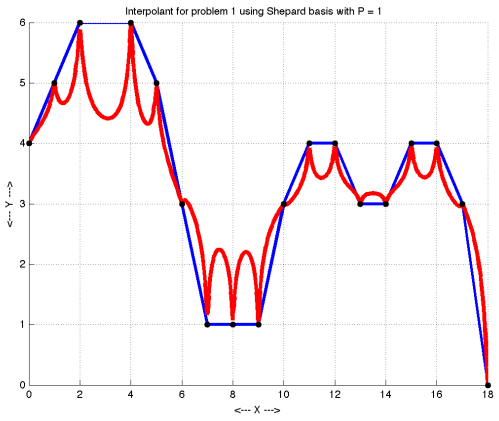
Setting p to 0 makes all data equally important, and the interpolant is a constant whose value is the data average.

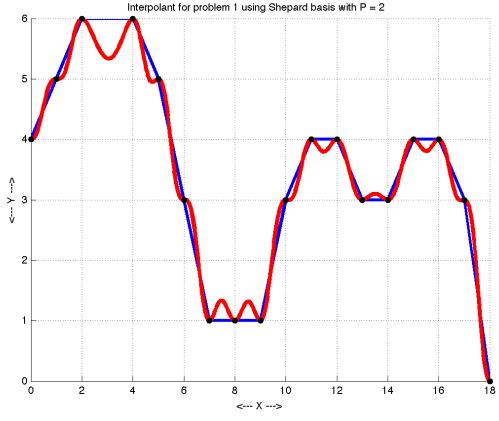
For an m-dimensional problem, it is typical to choose $m < p$ for best results. Low values of p yield a smoother interpolant, and high values tend to a "tiled" interpolant in which each data value dominates in its neighborhood.

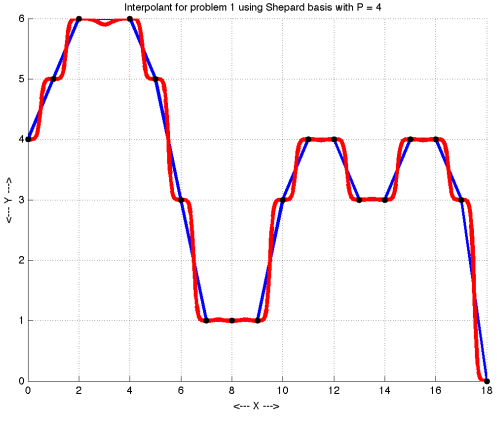
SHEP: Shepard Experiment 1

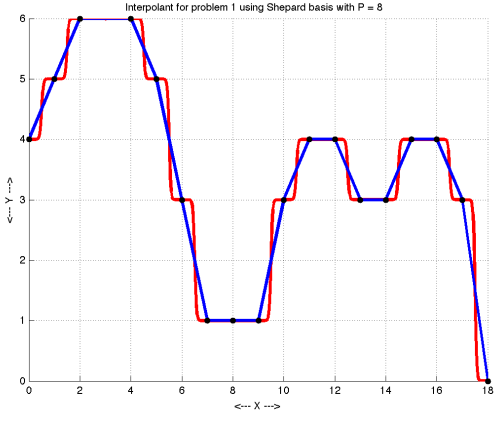


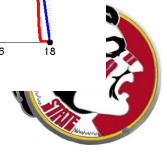




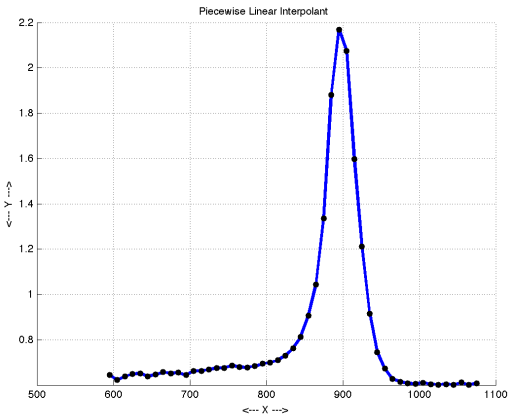


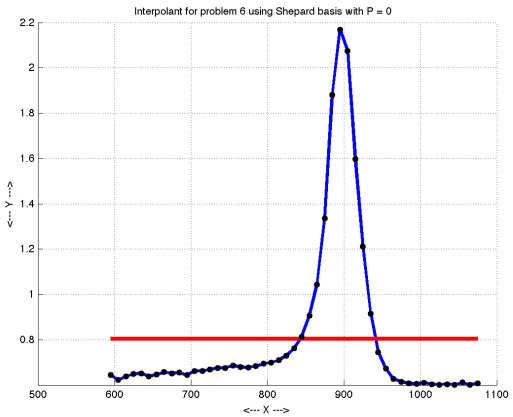


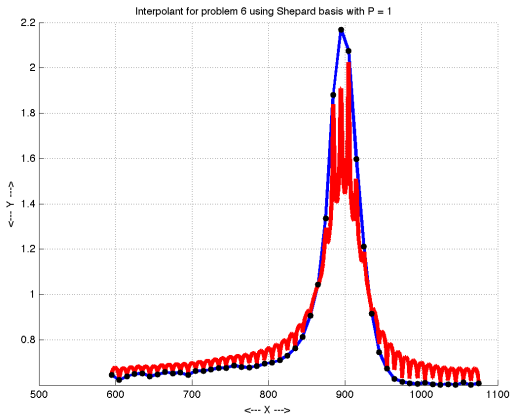


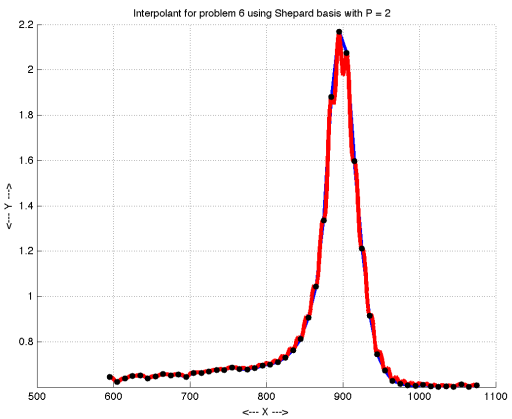


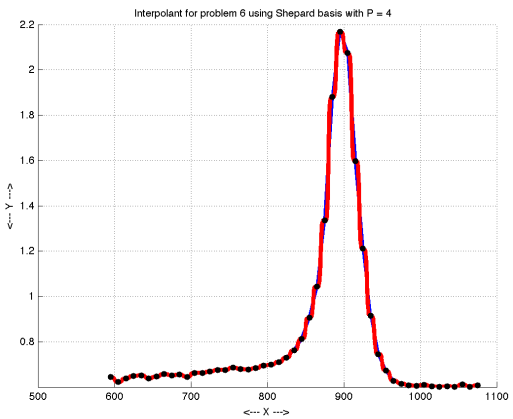
SHEP: Shepard Experiment 6

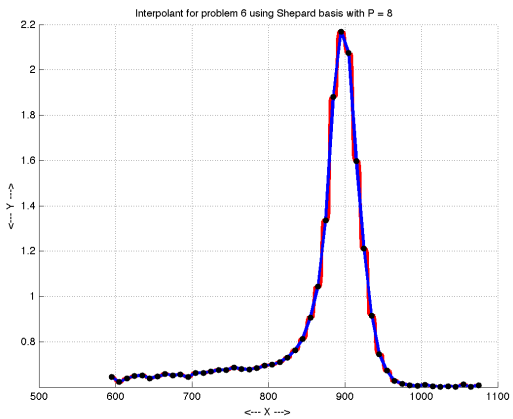




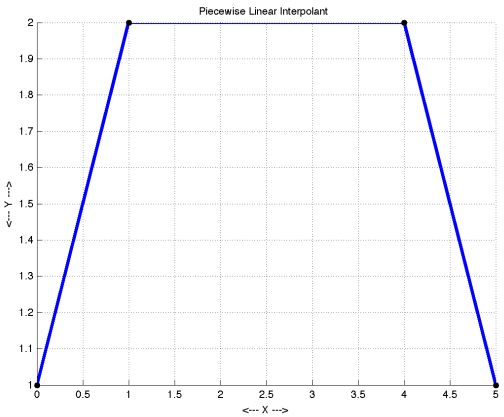


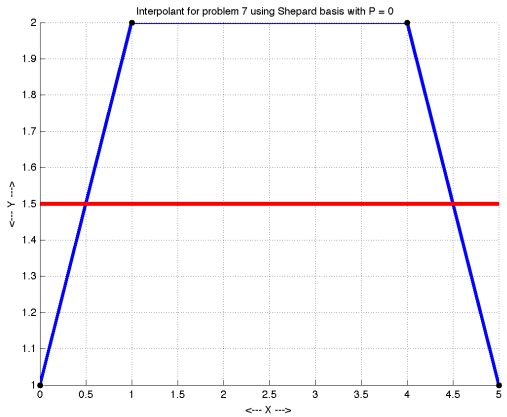


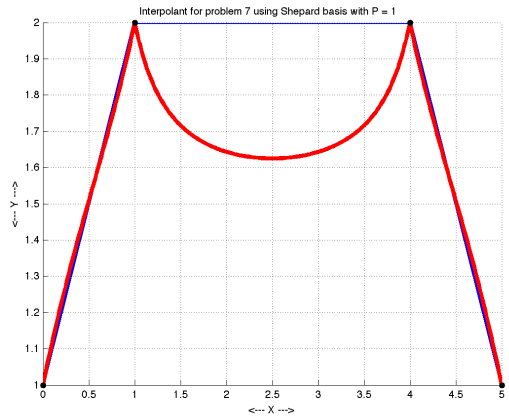


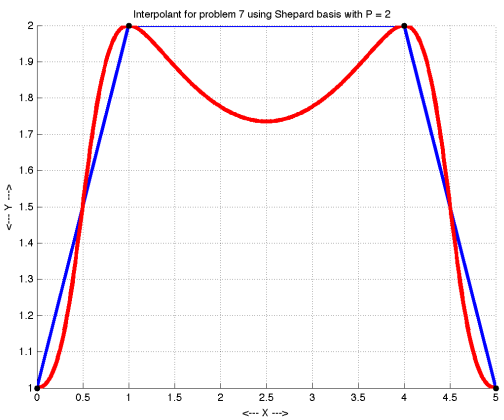


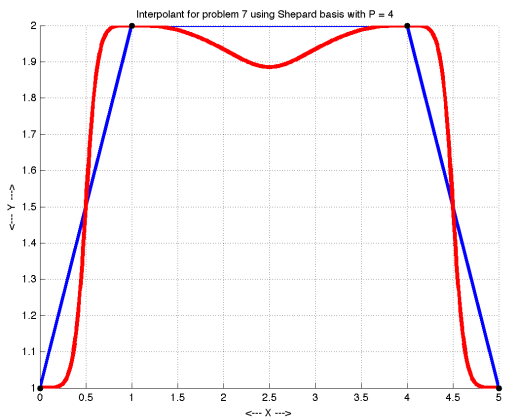
SHEP: Shepard Experiment 7

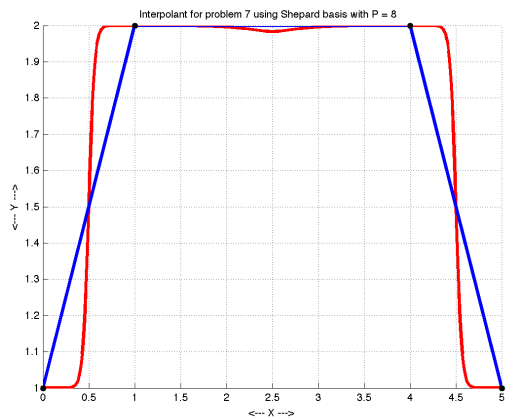


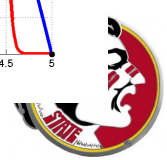












SHEP: The Weights

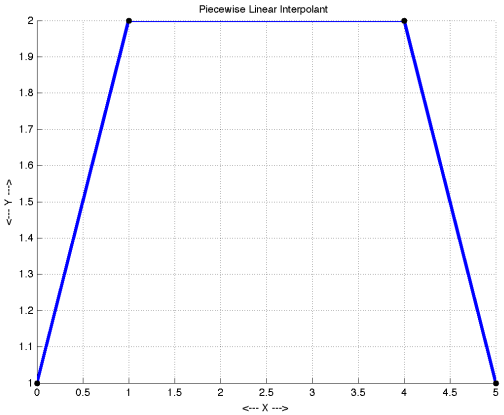
The weights $\hat{w}_i(x)$ are actually basis functions. They show up in the Shepard interpolation formula in the same way that the basis functions $l(i; x)$ show up in the Lagrange formula.

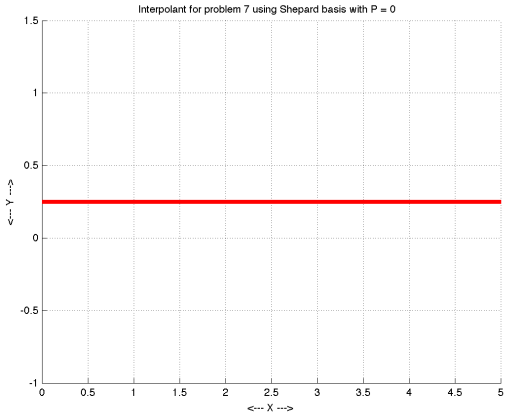
The interpolation formula only needs to know the distance between x and each data value x_i. This means that the formula works in the same way for any dimension.

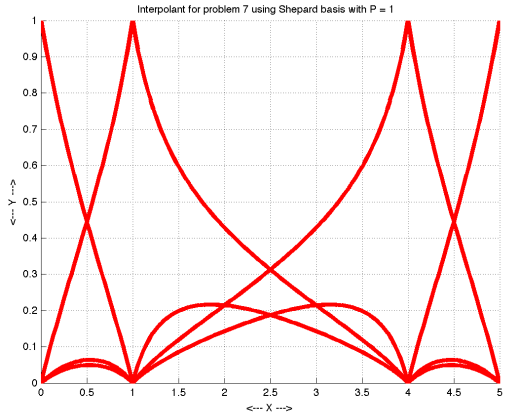
Thus, the cost is easy to calculate. Each evaluation of the interpolant costs us n weight calculations.

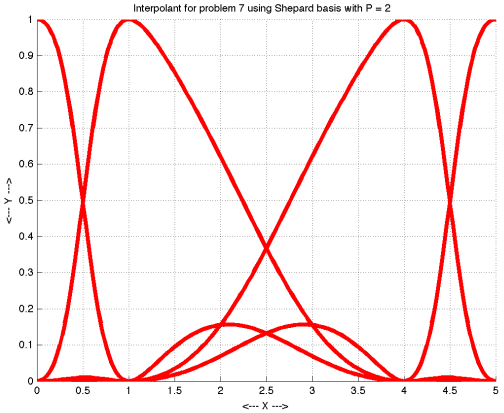
SHEP: Shepard Basis for Experiment 7

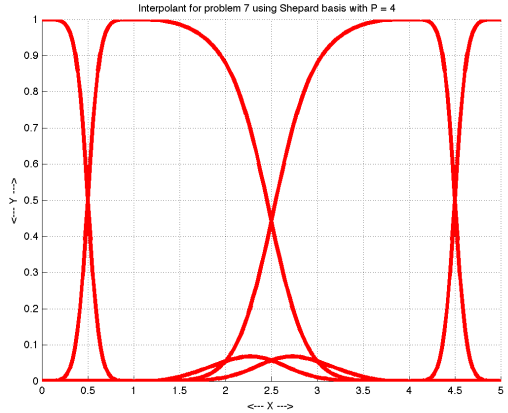
data	p = 0	p = 1
p = 2	p = 4	p = 8

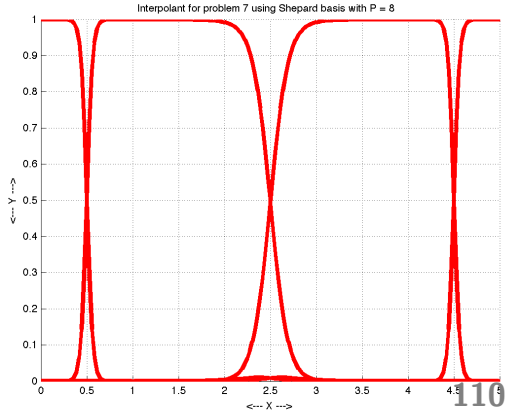












The Interpolation Problem in 1D

- Introduction
- Interpolation in 1D
- Nearest Interpolation
- Piecewise Linear Interpolation
- Vandermonde Interpolation
- Vandermonde Approximation
- Lagrange Interpolation: Even Nodes
- Lagrange Interpolation: Chebyshev Nodes
- Lagrange Approximation
- Barycentric Lagrange Interpolation
- Shepard Interpolation
- **Radial Basis Functions**

RBF: Radial Basis Functions

The Shepard approach used basis functions $\hat{w}_i(x)$, derived from the inverse of the distance $r(x_i; x) = ||x - x_i||$ to the various data points.

It is simple to generalize the Shepard interpolant by the use of a family of radial basis functions $\phi(r)$ to control the relative influence of the data values at arbitrary points in the region:

- multiquadric, $\phi_1(r) = \sqrt{r^2 + r_0^2}$;
- inverse multiquadric, $\phi_2(r) = \frac{1}{\sqrt{r^2 + r_0^2}}$;
- thin plate spline, $\phi_3(r) = r^2 \log(\frac{r}{r_0})$;
- gaussian, $\phi_4(r) = e^{-0.5r^2/r_0^2}$;

Here, r_0 is a scale factor parameter that might be a small multiple of the typical distance between data locations.

RBF: Radial Basis Functions

Given n multidimensional data points (x_i, y_i), and having chosen a radial basis function $\phi(r)$, the form of the RBF interpolant is:

$$g(x) = \sum_{j=1}^{n} w_j * \phi(||x - x_j||)$$

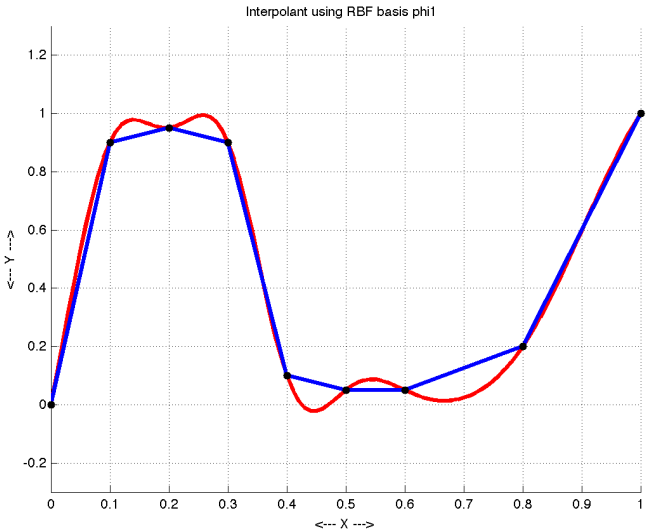
where the weights w are determined by solving (the dense linear system!):

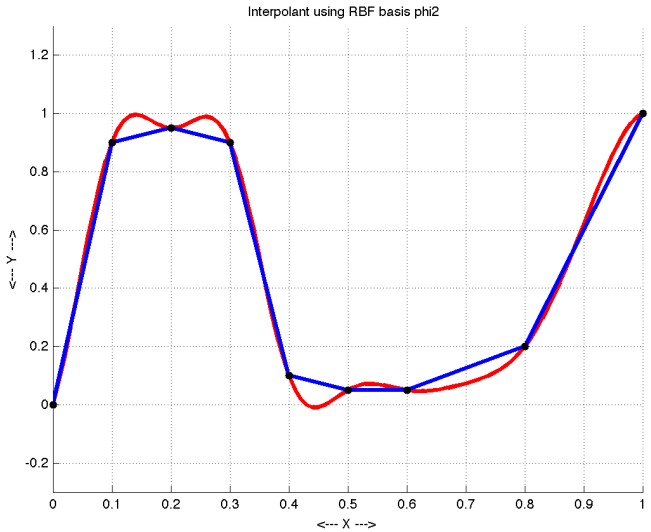
$$g(x_i) = \sum_{i=j}^{n} w_j * \phi(||x_i - x_j||) = y_i, \quad i = 1, ..., n$$

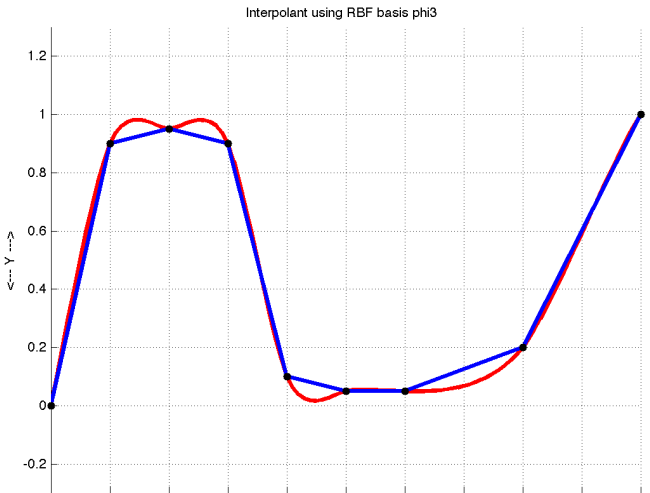
which is simply the usual interpolation equations.

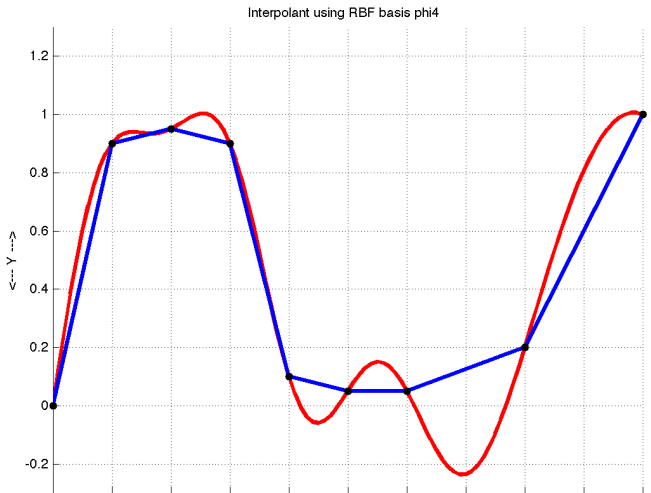
RBF: Experiment 5, R0 = 1/8

`phi1(x)` | `phi2(x)`
`phi3(x)` | `phi4(x)`



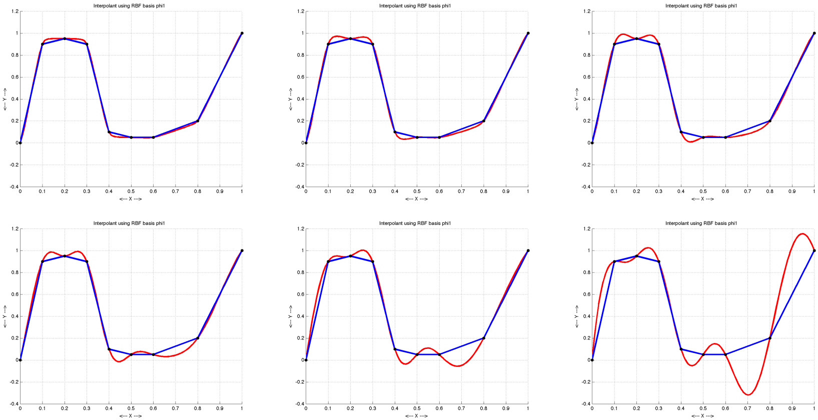






RBF: Experiment 5, Using PHI1

```
r0 = 0.0125          |     r0 = 0.0250          |     r0 = 0.0500
r0 = 0.1000          |     r0 = 0.2000          |     r0 = 0.4000
```



Conclusions:

Conclusions:

A bad selection of data locations $\{x_i\}$ *can doom the process;*

If the data is simply given to us, we need to be concerned if there are data locations that are very close.

If the data is evenly spaced or uniformly distributed over a finite region, there can be a tendency for oscillations and other effects to arise at points far from the center. may are likely to

If we can select the data, then the Chebyshev distribution can be useful.

If we need to sample a growing sequence of data, we can use a PDF related to the Chebyshev distribution.

Conclusions:

A bad representation of the interpolating space can cause unbounded error;

The "natural" basis for the space of polynomials is not useful for interpolation.

The same problem arises in higher dimensions. This depends mainly on the degree of the polynomials, not the number of interpolation points. In higher dimensions, the degree rises more slowly, so we may not see the problem quite so soon.

Conclusions:

A function may interpolate your data and do almost anything elsewhere.

Every one of the radial basis function interpolants interpolates the data. And yet for small values of the parameter r_0, the interpolant is essentially zero except very near the data.

Similarly, polynomial interpolants of high degree will begin to oscillate wildly in between the interpolation points (and for high enough degree, they won't even interpolate.)

Conclusions:

High degree polynomials can do a good interpolation job.

When we have a lot of data, it makes sense to use it all.

To avoid oscillations, we could try a Lagrange approximant, although this requires solving a linear system.

If we can select the locations of the data points, then the barycentric Lagrange interpolant gives us an explicit, stable, and efficient representation of the interpolating polynomial, even in cases where the degree is 1000.

Conclusions:

The polynomial interpolant for the Vandermonde basis almost guarantees an ill-conditioned system.

Particularly if the data is specified in advance, (can't select the data locations) your interpolant may be inaccurate.

Checking that the resulting interpolant simply interpolates the data gives no guarantee about what is happening at the non-data locations.

If you can select your data points, and use a Lagrange basis, and evaluate the interpolant using the barycentric formula, you may get better results than folklore suggests.

Conclusions:

These observations for the 1D problem will be useful for higher dimensions, particularly in the case of product rules, and of sparse grid rules, which can be constructed from 1d rules, and which inherit many of their properties.

Reference:

Jean-Paul Berrut, Lloyd Trefethen,
Barycentric Lagrange Interpolation,
SIAM Review,
Volume 46, Number 3, September 2004, pages 501-517.

The Interpolation Problem in 1D