Radial basis function network

In[906]:=
```
x = 0; Remove["Global`*"]; DateList[Date[]] // Most
```

Out[906]=
```
{2020, 5, 11, 20, 56}
```

Radial basis functions have grown in popularity, particularly for multivariate interpolation with scattered data.

The key property is that a unique interpolant ALWAYS exists.

This is not true of any polynomial basis interpolation.

- Infinitely Smooth RBFs

These radial basis functions are from $C^{\infty}(\mathbb{R})$ and are strictly positive definite functions[12] that r

- Gaussian:

$$\varphi(r) = e^{-(\varepsilon r)^2}$$

- Multiquadric:

$$\varphi(r) = \sqrt{1 + (\varepsilon r)^2}$$

- Inverse quadratic:

$$\varphi(r) = \frac{1}{1 + (\varepsilon r)^2}$$

- Inverse multiquadric:

$$\varphi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$$

- Polyharmonic spline:

$$\begin{aligned} \varphi(r) &= r^k, & k &= 1, 3, 5, \dots \\ \varphi(r) &= r^k \ln(r), & k &= 2, 4, 6, \dots \end{aligned}$$

For even-degree polyharmonic splines $(k = 2, 4, 6, \dots)$, to avoid numerical problems at computational implementation is often written as $\varphi(r) = r^{k-1} \ln(r^r)$.

- Thin plate spline (a special polyharmonic spline):

$$\varphi(r) = r^2 \ln(r)$$

- Compactly Supported RBFs

These RBFs are compactly supported and thus are non-zero only within a radius of $1/\varepsilon$, and t matrices

- Bump function:

$$\varphi(r) = \begin{cases} \exp\left(-\frac{1}{1-(\varepsilon r)^2}\right) & \text{for } r < \frac{1}{\varepsilon} \\ 0 & \text{otherwise} \end{cases}$$

Define radial basis function (rbf)

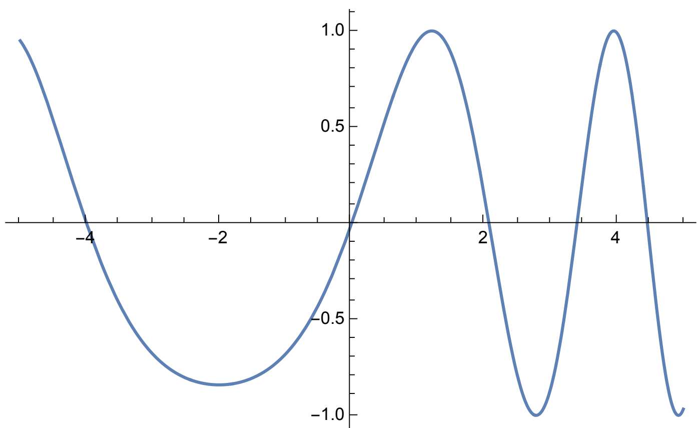
G will be our node function. We make it evaluate vectors elementwise.
The parameter c is the center of the rbf, and r scales the rbf

In[907]:=
```
Clear[G]
G[x_] := Exp[-(ϵ x)^2]
```

Create data set

In[909]:=
```
f[x_] = Sin[ x x / 4 + x];
Plot[f[x], {x, -5, 5}, PlotRange → All]
```

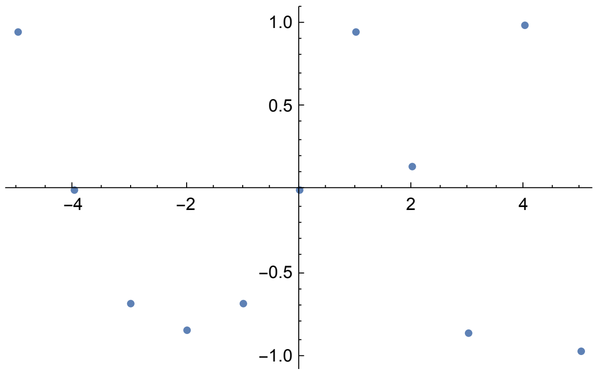
Out[910]=



We shall interpolate this function using 11 uniformly distributed points in [-5,5]

In[911]:=
```
{xmin, xmax} = {-5., 5.}; xpts = Range[-5, 5, 1];
npts = Length[xpts];
ypts = f /@ xpts; data = {xpts, ypts} // N // Transpose;
ListPlot[data]
```

Out[914]=



Define the model

Since we interpolate, the centers will be the data points. and the basis will be the rbfs with those centers

In[915]:=
```
centers = xpts;
```

In[916]:=
```
basis = Table[G[x - c], {c, centers}]
```

Out[916]=
$$\left\{e^{-(5+x)^2 \epsilon^2}, e^{-(4+x)^2 \epsilon^2}, e^{-(3+x)^2 \epsilon^2}, e^{-(2+x)^2 \epsilon^2}, e^{-(1+x)^2 \epsilon^2}, e^{-x^2 \epsilon^2}, e^{-(-1+x)^2 \epsilon^2}, e^{-(-2+x)^2 \epsilon^2}, e^{-(-3+x)^2 \epsilon^2}, e^{-(-4+x)^2 \epsilon^2}, e^{-(-5+x)^2 \epsilon^2}\right\}$$

The model function will be a linearly weighted sum of the outputs of the rbfs

In[917]:=
```
avec = Table[a_i, {i, 1, npts}];
model = avec.basis
```

Out[918]=
$$e^{-(5+x)^2 \epsilon^2} a_1 + e^{-(4+x)^2 \epsilon^2} a_2 + e^{-(3+x)^2 \epsilon^2} a_3 + e^{-(2+x)^2 \epsilon^2} a_4 + e^{-(1+x)^2 \epsilon^2} a_5 + e^{-x^2 \epsilon^2} a_6 + e^{-(-1+x)^2 \epsilon^2} a_7 + e^{-(-2+x)^2 \epsilon^2} a_8 + e^{-(-3+x)^2 \epsilon^2} a_9 + e^{-(-4+x)^2 \epsilon^2} a_{10} + e^{-(-5+x)^2 \epsilon^2} a_{11}$$

Define the set of variables and make 1 the initial guess for all

In[919]:=
```
vars = avec; init = vars - vars + 1;
varsin = {vars, init} // Transpose;
```

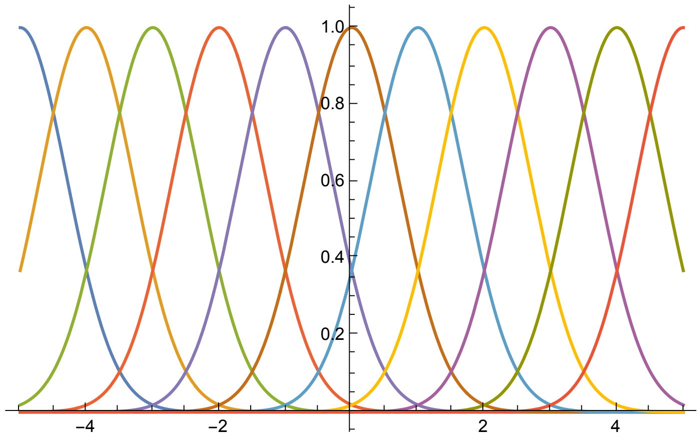
Interpolation

We shall interpolate. The number of data points equals the number of free parameters, a_i. We have not chosen the tuning parameter. We will try three different values.

Fit our example: $\epsilon = 1$

In[967]:=
```
ϵ = 1;
Plot[basis, {x, xmin, xmax}]
```

Out[968]=



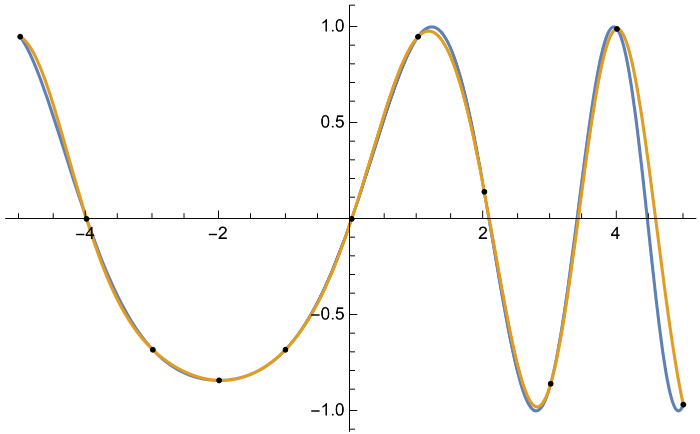
Compute the inner products of each pair to ascertain the covariance. Since all eigenvalues are close, these functions are relatively independent.

In[969]:=
```
Table[NIntegrate[basis[[i]] × basis[[j]], {x, xmin, xmax}],
  {i, 1, npts}, {j, 1, npts}];
Eigenvalues[
 %]
```

Out[970]=
```
{3.01918, 2.68023, 2.19879, 1.6681, 1.17125,
 0.761839, 0.459511, 0.257726, 0.136445, 0.0734326, 0.0495276}
```

In[971]:=
```
fit = FindFit[data, model, varsin, x, MaxIterations → 500];
modelf = Function[{x}, Evaluate[model /. fit]];
Plot[{f[x], modelf[x]}, {x, xmin, xmax}, PlotRange → All, Epilog → Map[Point, data]]
```

Out[973]=



Nice fit.

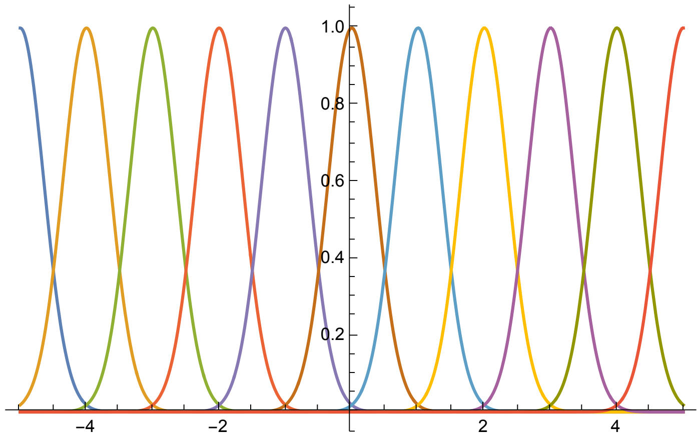
In[974]:=
```
NIntegrate[(f[x] - modelf[x])^2, {x, xmin, xmax}]^(1/2)
```

Out[974]= 0.286344

Fit our example: $\epsilon = 2$ (more spiky rbfs)

```
ϵ = 2;
Plot[basis, {x, xmin, xmax}, PlotRange → All]
```

Out[976]=



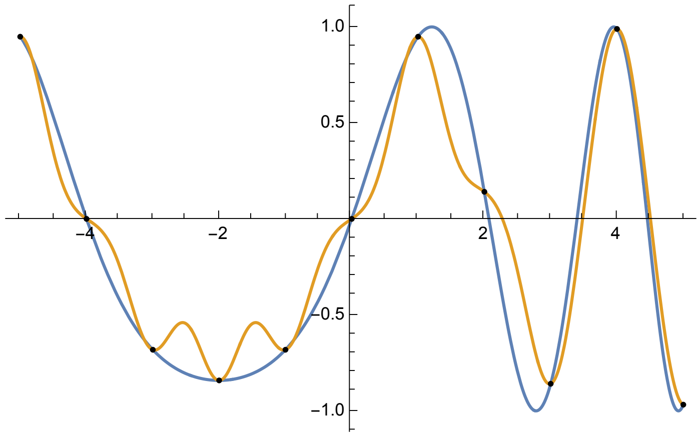
In[977]:=

```
Table[NIntegrate[basis[[i]] × basis[[j]], {x, xmin, xmax}],
   {i, 1, npts}, {j, 1, npts}];
Eigenvalues[
 %]
```

Out[978]= {0.78895, 0.766387, 0.730967, 0.685775, 0.634769, 0.582498, 0.533825, 0.493704, 0.466931, 0.291363, 0.291362}

In[979]:=
```
fit = FindFit[data, model, varsin, x, MaxIterations → 500];
modelf = Function[{x}, Evaluate[model /. fit]];
Plot[{f[x], modelf[x]}, {x, xmin, xmax}, PlotRange → All, Epilog → Map[Point, data]]
```

Out[981]=



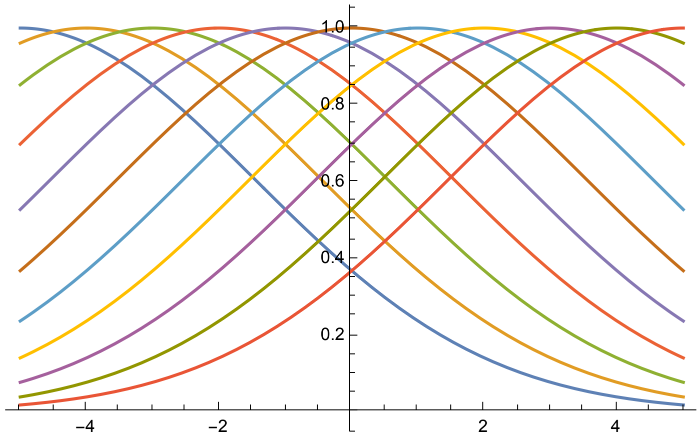
In[982]:=
```
NIntegrate[(f[x] - modelf[x])^2, {x, xmin, xmax}]^(1/2)
```

Out[982]= 0.629905

Fit our example: ϵ=1/5 (flatter rbfs)

```
ϵ = 1 / 5;
Plot[basis, {x, xmin, xmax}, PlotRange → All]
```

Out[984]=

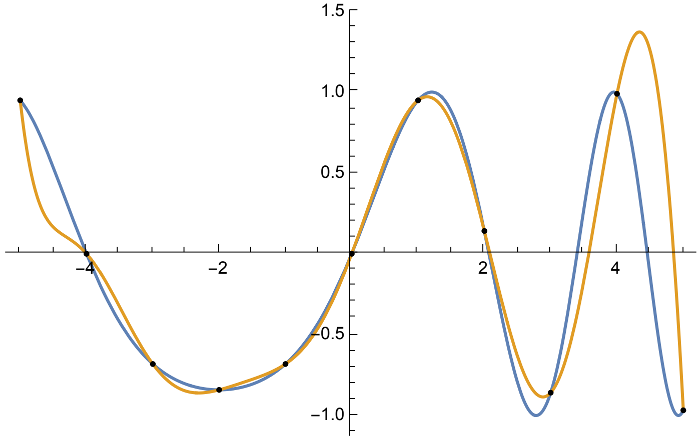



```
In[985]:= Table[NIntegrate[basis[[i]] × basis[[j]], {x, xmin, xmax}],
   {i, 1, npts}, {j, 1, npts}];
Eigenvalues[
 %]
```

Out[986]= $\{44.2055, 8.30632, 0.599869, 0.0199102, 0.000354803, 3.75935\times10^{-6}, 2.49712\times10^{-8}, 1.05482\times10^{-10}, 2.75658\times10^{-13}, -1.09776\times10^{-15}, 9.75085\times10^{-16}\}$

In[987]:=
```
fit = FindFit[data, model, varsin, x, MaxIterations → 500];
modelf = Function[{x}, Evaluate[model /. fit]];
Plot[{f[x], modelf[x]}, {x, xmin, xmax}, PlotRange → All, Epilog → Map[Point, data]]
```

Out[989]=



In[952]:= $\text{NIntegrate}\left[(f[x]-\text{modelf}[x])^2, \{x, \text{xmin}, \text{xmax}\}\right]^{1/2}$

Out[952]= 1.13886

Comments

RBFs are more useful in multiple dimensions, particularly when the data is scattered.

Each RBF represents a data point and interpolation allows us to approximate a function between the data points.

Choosing the tuning parameter is a matter of “art”. One can try different values and then use out-of-sample information to choose a good value.

As the the size of data sets increases, the use of flatter RBFs is good EXCEPT for the ill-conditioning problems.