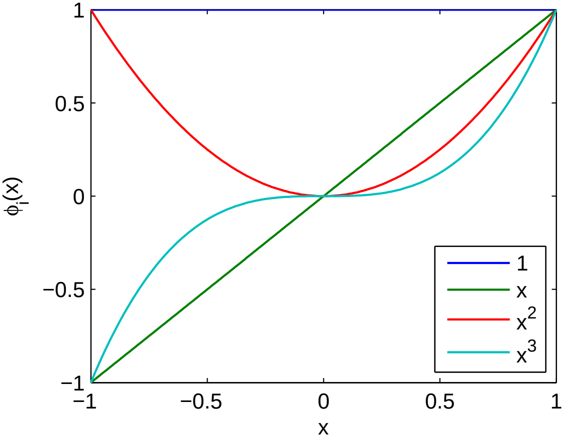
# Constructing explicit feature vectors

- Polynomial features (max degree $d$)

Special case, n=1: $$\phi(z) = \begin{bmatrix} z^d \\ z^{d-1} \\ \vdots \\ z \\ 1 \end{bmatrix} \in \mathbb{R}^{d+1}$$

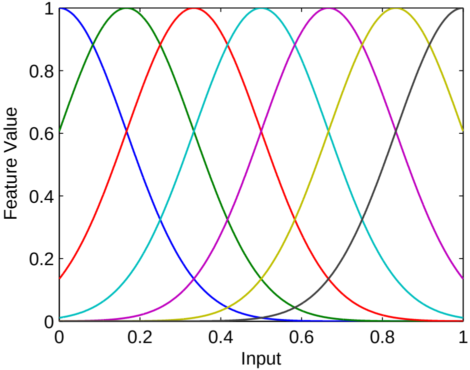
General case: $$\phi(z) = \left\{ \prod_{i=1}^{n} z_i^{b_i} : \sum_{i=1}^{n} b_i \leq d \right\} \in \mathbb{R}^{\binom{n+d}{n}}$$




Plot of polynomial bases

- Radial basis function (RBF) features
  - Defined by bandwidth $\sigma$ and $k$ RBF centers $\mu_j \in \mathbb{R}^n$, $j = 1, \ldots, k$

$$\phi_j(z) = \exp\left\{\frac{-\|z-\mu_j\|^2}{2\sigma^2}\right\}$$



# Difficulties with non-linear features

- Problem #1: Computational difficulties
  - Polynomial features,

$$k = \binom{n+d}{d} = O(d^n)$$

  - RBF features; suppose we want centers in uniform grid over input space (w/ $d$ centers along each dimension)

$$k = d^n$$

  - In both cases, exponential in the size of the input dimension; quickly intractable to even store in memory

- Problem #2: Representational difficulties
  - With many features, our prediction function becomes very expressive
  - Can lead to *overfitting* (low error on input data points, but high error nearby)

- A few ways to deal with representational problem:
  - Choose less expressive function (e.g., lower degree polynomial, fewer RBF centers, larger RBF bandwidth)
  - *Regularization*: penalize large parameters $\theta$

    $$\operatorname*{minimize}_{\theta} \sum_{i=1}^{m} \ell(\hat{y}_i, y_i) + \color{red}{\lambda \|\theta\|_2^2}$$

    $\lambda$: regularization parameter, trades off between low loss and small values of $\theta$ (often, don't regularize constant term)
  - We'll come back to this issue when talking about evaluating machine learning methods

# Implicit feature vectors (kernels)

- One of the main trends in machine learning in the past 15 years
- Kernels let us work in high-dimensional feature spaces without explicitly constructing the feature vector
- This addresses the first problem, the computational difficulty

- Simple example, polynomial feature, $n = 2$, $d = 2$

$$\phi(z) = \begin{bmatrix} 1 \\ \sqrt{2}z_1 \\ \sqrt{2}z_2 \\ z_1^2 \\ \sqrt{2}z_1 z_2 \\ z_2^2 \end{bmatrix}$$

- Let's look at the *inner product* between two different feature vectors

$$\begin{aligned} \phi(z)^T \phi(z') &= 1 + 2z_1 z_1' + 2z_2 z_2' + z_1^2 {z_1'}^2 + 2z_1 z_2 z_1' z_2' + z_2^2 {z_2'}^2 \\ &= 1 + 2(z_1 z_1' + z_2 z_2') + (z_1 z_1' + z_2 z_2')^2 \\ &= 1 + 2(z^T z') + (z^T z')^2 \\ &= (1 + z^T z')^2 \end{aligned}$$

- General case: $(1 + z^T z')^d$ is the inner product between two polynomial feature vectors of max degree $d$ ($\binom{n+d}{d}$-dimensional)
  - But, can be computed in only $O(n)$ time

- We use the notation of a *kernel function* $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ that computes these inner products

$$K(z, z') = \phi(z)^T \phi(z')$$

- Some common kernels

$$\text{polynomial (degree } d\text{):} \quad K(z, z') = (1 + z^T z')^d$$

$$\text{Gaussian (bandwidth } \sigma\text{):} \quad K(z, z') = \exp\left\{\frac{-\|z - z'\|_2^2}{2\sigma^2}\right\}$$

# Using kernels in optimization

- We can compute inner produucts, but how does this help us solve optimization problems?

- Consider (regularized) optimization problem we've been using

$$\underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \ell(\theta^T \phi(x_i), y_i) + \lambda \|\theta\|_2^2$$

- *Representer theorem*: The solution to above problem is given by

$$\theta^\star = \sum_{i=1}^{m} \alpha_i \phi(x_i), \quad \text{for some } \alpha \in \mathbb{R}^m, \quad (\text{or } \theta^\star = \Phi^T \alpha)$$

- Notice that

$$(\Phi\Phi^T)_{ij} = \phi(x_i)^T\phi(x_j) = K(x_i, x_j)$$

- Abusing notation a bit, we'll define the *kernel matrix* $K \in \mathbb{R}^{m\times m}$

$$K = \Phi\Phi^T, \quad (K_{ij} = K(x_i, x_j))$$

can be computed *without* constructing feature vectors or $\Phi$

- Let's take (reguarlized) least squares objective...

$$J(\theta) = \|\Phi\theta - y\|_2^2 + \lambda\|\theta\|_2^2$$

- and substitute $\theta = \Phi^T\alpha$

$$\begin{aligned} J(\alpha) &= \|\Phi\Phi^T\alpha - y\|_2^2 + \lambda\alpha^T\Phi\Phi^T\alpha \\ &= \|K\alpha - y\|_2^2 + \lambda\alpha^T K\alpha \\ &= \alpha^T KK\alpha - 2y^T K\alpha + y^T y + \lambda\alpha^T K\alpha \end{aligned}$$

- Taking the gradient w.r.t. $\alpha$ and setting to zero

$$\nabla_\alpha J(\alpha) = 2KK\alpha - 2Ky + 2\lambda K\alpha \Rightarrow \alpha^\star = (K + \lambda I)^{-1}y$$

- How do we compute prediction on a new input $x'$?

$$\hat{y}' = \theta^T \phi(x') = \left( \sum_{i=1}^{m} \alpha_i \phi(x_i) \right)^T \phi(x') = \sum_{i=1}^{m} \alpha_i K(x_i, x')$$

- Need to keep around all examples $x_i$ in order to make a prediction; *non-parametric* method

- MATLAB code for polynomial kernel

```
% computing alphas
d = 6;
lambda = 1;
K = (1 + X*X').^d;
alpha = (K + lambda*eye(m)) \ y;

% computing prediction
k_test = (1 + x_test*X').^d;
y_hat = k_test*alpha;
```

- Gaussian kernel

```
sigma = 0.1;
lambda = 1;
K = exp(-0.5*sqdist(X', X')/sigma^2)
alpha = (K + lambda*eye(m)) \ y;
```