

# Lecture on Numerical Methods

April 5, 2009

# Objectives

- Linear/Non-linear equations
- Optimization
- Differentiation/Integration
- Functional approximation

# COMPECON Library

- <http://www4.ncsu.edu/~pfackler/compecon/toolbox.html>
- Miranda and Fackler '02
- Implement most numerical tools I describe here
- Written in MATLAB with C-files for bottlenecks
- Also (very) useful resources:
  - Judd '1998
  - Press et. al. 1992: [www.nr.com](http://www.nr.com): routines Fortran and C

## Linear Equations: $Ax = b$

- Most non-linear problems reduce to linear equations
- $x = A \setminus b$  in MATLAB (also least-squares solver)
- May run into rounding problems. (similar to multi-collinearity)

## Non-linear equations. $f(x) = 0$

- Bisection (**bisect** in Compecon)
  - 1-dimensional problems. Need continuity
  - Choose  $a_0, b_0$ , s.t.  $\text{sgn } f(a_0) \neq \text{sgn } f(b_0)$ . Let  $c_0 = \frac{a_0 + b_0}{2}$ 
    - Set  $a_1 = a_0$  and  $b_1 = c_0$  if  $\text{sgn } f(b_1) = \text{sgn } f(c_0)$
    - Set  $a_1 = c_0$  and  $b_1 = b_0$  if  $\text{sgn } f(b_1) \neq \text{sgn } f(c_0)$
    - Continue until interval  $< \epsilon$
  - Converges in  $\log\left(\frac{b-a}{\epsilon}\right)$  iterations. E.g., 10 to  $10^{-7}$  in 27 it.

## Non-linear equations. $f(x) = 0$

- Newton's method (**newton** in Compecon)

$$x^{i+1} = x^i - \left[ \frac{\partial f(x^i)}{\partial x} \right]^{-1} f(x^i)$$

- Need Jacobian info
  - can oscillate/diverge if  $f'$  changes sign too often
  - but very fast if it works
- Broyden's method (**broyden** in Compecon)

$$x^{i+1} = x^i - (A^i)^{-1} f(x^i)$$

$$A^{i+1} = A^i + [f(x^{i+1}) - f(x^i) - A^i(x^{i+1} - x^i)] \frac{(x^{i+1} - x^i)'}{(x^{i+1} - x^i)'(x^{i+1} - x^i)}$$

## Convergence rates

- Let  $x^i \in R^n : \lim_{i \rightarrow \infty} x^i = \bar{x}$
- $x^i$  converges at rate  $q$  if  $\lim_{i \rightarrow \infty} \frac{\|x^{i+1} - \bar{x}\|}{\|x^i - \bar{x}\|^q} < \infty$ 
  - E.g.,  $q = 2$  (quadratic):  $\frac{1}{2}, \frac{1}{4}, \frac{1}{16}, \frac{1}{256}, \dots, \frac{1}{2^{2^k}}$
- $x^i$  converges linearly at rate  $\beta$  if  $\lim_{i \rightarrow \infty} \frac{\|x^{i+1} - \bar{x}\|}{\|x^i - \bar{x}\|} < \beta < 1$ 
  - E.g.,  $\beta = \frac{1}{2} : \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \dots, \frac{1}{2^i}$
- Bisection: linear with  $\beta = \frac{1}{2}$
- Function iteration: linear with  $\beta = \left\| \frac{\partial f(\bar{x})}{\partial x} \right\|_{\infty}$ .
- Newton: quadratic
- Broyden: superlinear ( $q \in (1, 2)$ )

## Optimization: $\max_x f(x)$ where $f : R^n \rightarrow R$

- Golden search (**golden** in Compecon)
  - Searches for maxima on  $[a, b]$  in 1-dimension, ( $\approx$  bisection)
  - Given  $x_1 < x_2 \in [a, b]$ , replace  $[a, b]$  with  $[a, x_2]$  if  $f(x_1) > f(x_2)$  or with  $[x_1, b]$  if  $f(x_1) < f(x_2)$
  - Slow but sure. I use it frequently
- Nelder-Mead (simplex): (**neldmead** in Compecon)
  - Evaluate  $f : R^n \rightarrow R$  at  $n + 1$  points (simplex in  $n$  dimensions.)
  - Reflect simplex away from point with lowest function value
  - Slow and unreliable

## Optimization: $\max_x f(x)$ where $f : R^n \rightarrow R$

- Quasi-Newton (**qnewton** in Compecon)

- Variants of Newton update:

$$x^{i+1} = x^i - \left[ \frac{\partial^2 f(x)}{\partial x \partial x'} \right]^{-1} \frac{\partial f(x)}{\partial x}$$

- Hessian may be expensive, not negative-definite everywhere
- Replace with negative-definite approximations:
  1. identity matrix (steepest ascent)
  2. Davidson-Fletcher-Powell
  3. Broyden-Fletcher-Goldfarb-Shano

# Differentiation

- `fdjac` and `fdhess` in Compecon

$$f(x + h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3)$$

$$f(x - h) = f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 + O(h^3)$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

- $h = \sqrt{\text{eps}}$

# Integration

- Focus on Gaussian quadrature
  - Newton-Cotes (simpson and trapezoid) also in Compecon
  - based on local approximations of  $f$  with line or cubic
  - `help qnwsimp` and `help qnwtrap`
  - Also Monte-Carlo methods available
- Gaussian quadrature: computed relative to a particular weighting function  $w(x)$ . Used to approximate

$$\int_a^b f(x)w(x)dx$$

- Choose  $n$  nodes  $x_1, \dots, x_n$  and weights  $\omega_1, \dots, \omega_n$  to satisfy  $2n$  moment-matching conditions:

$$\int x^k w(x)dx = \sum_{i=1}^n \omega_i x_i^k \text{ for } k = 0, \dots, 2n - 1$$

## Integration

- Choose  $n$  nodes  $x_1, \dots, x_n$  and weights  $\omega_1, \dots, \omega_n$  to satisfy  $2n$  moment-matching conditions:

$$\int x^k w(x) dx = \sum_{i=1}^n \omega_i x_i^k \text{ for } k = 0, \dots, 2n - 1$$

- One interpretation: replace complicated pdf  $w(x)$  with discrete distribution with  $\omega_i$  and mass points  $x_i$
- Nasty numerical problem
- Specialized routines/tables/starting guesses for standard weighting functions available
- Integral approximated with  $\int f(x)w(x)dx \approx \sum_{i=1}^n \omega_i f(x_i)$

# Integration

- Weighting functions coded in Compecon
  - Gauss-Legendre (`qnwlege`):  $w(x) = 1$
  - Gauss-Chebyshev (`qnwcheb`):  $w(x) = (1 - x^2)^{-\frac{1}{2}}$ 
    - Weights/nodes in closed form. Can change variable to integrate any  $f(x)$ . See Judd (1998)
  - Gaussian pdf: (`qnwnorm`)
  - Uniform pdf: (`qnwunif`)
  - Beta pdf: (`qnwbeta`)
  - Gamma pdf: (`qnwgamma`)
- Extension to  $>1$  dimension straightforward
  - E.g.,  $\int_{a_2}^{b_2} \int_{a_1}^{b_1} f(x_1, x_2) dx_1 dx_2 \approx \sum_{i=1}^{n_2} \sum_{i=1}^{n_1} w_{1i} w_{2i} f(x_1, x_2)$

# Integration

- Example

Expected value of  $\frac{x}{\exp(y)}$ , where  $x \sim N(.1, .2)$ ,  $y \sim N(.2, .3)$

```
n=[7,7]; /* nodes in each dimension */  
varcov=[.2,0;0,.3]; /* variance-covariance of x,y */  
mean=[.1,.2]; /* mean of x,y */  
[x,w]=qjwnorm(n,mean,varcov); /* nodes and weights */  
Ev=w'*(x(:,1)./exp(x(:,2)));
```

## Function approximation: $f : D \subset \mathbb{R}^k \rightarrow \mathbb{R}^m$

- Goal: approximate  $f$  with  $\hat{f}$  to minimize

$$\|f - \hat{f}\|_{\infty} = \sup_{x \in D} |f(x) - \hat{f}(x)|$$

- Weierstrass theorem: if  $f$  continuous,  $D$  closed and bounded,  $\exists$  polynomial  $p$  s.t.

$$\|f - p\|_{\infty} < \epsilon$$

- Let  $\hat{f}(\cdot; c)$  be linear combination of low-order polynomials
- Choose coefficients  $c$  on polynomials to minimize distance between  $\hat{f}$  and  $f$  at finite number of nodes in  $D$
- Choice of polynomials and nodes important

## Choice of polynomials

- Using  $\hat{f}(\cdot; c) = c_0 + c_1x + c_2x^2 + c_3x^3 + \dots$  BAD IDEA
- Let  $x = x_1, \dots, x_n$  be nodes for interpolation
- Choose  $c$  to ensure  $\hat{f}(x; c) = f(x)$

- $$\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^n \\ 1 & x_2 & x_2^2 & \dots & x_2^n \\ 1 & x_3 & x_3^2 & \dots & x_3^n \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \dots \\ c_n \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \dots \\ f(x_n) \end{pmatrix}$$

- Matrix badly ill-conditioned (multi-colinear), for large  $n$

## Choice of polynomials

- Choose orthogonal polynomials as basis functions
- Orthogonal w.r.t basis function  $w(x)$

$$\int_D p_n(x)p_m(x)w(x) = 0$$

- Chebyshev polynomials:  $w(x) = (1 - x^2)^{-\frac{1}{2}}$

$$\phi_1(x) = 1$$

$$\phi_2(x) = x$$

$$\phi_3(x) = 2x^2 - 1$$

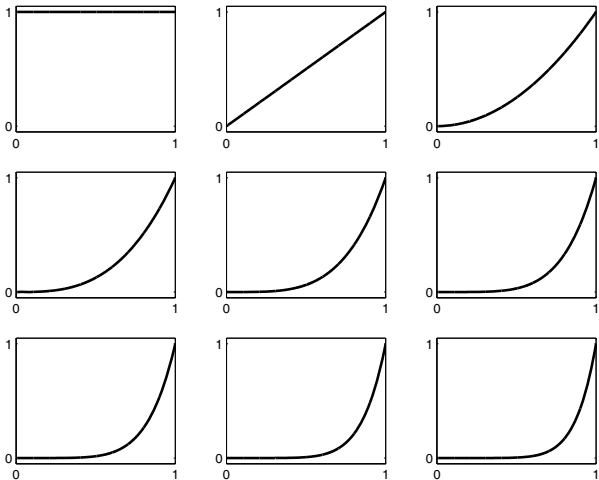
$$\phi_4(x) = 4x^3 - 3x$$

$$\phi_5(x) = 8x^4 - 8x^2 + 1$$

$$\phi_n(x) = 2x\phi_{n-1} - \phi_{n-2}$$

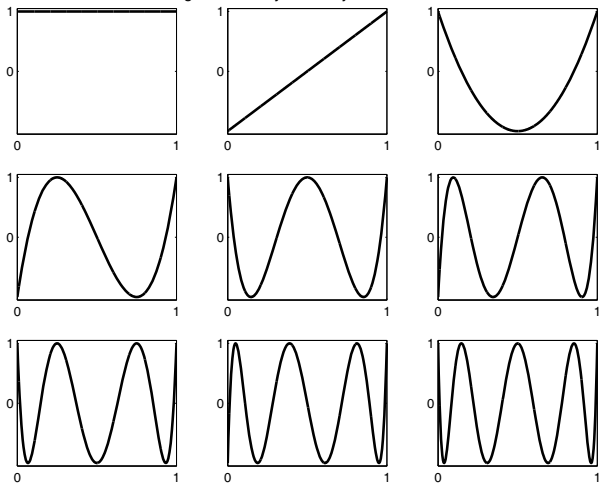
# Monomial basis functions

Figure 2: Monomial Basis Functions



# Chebyshev basis functions

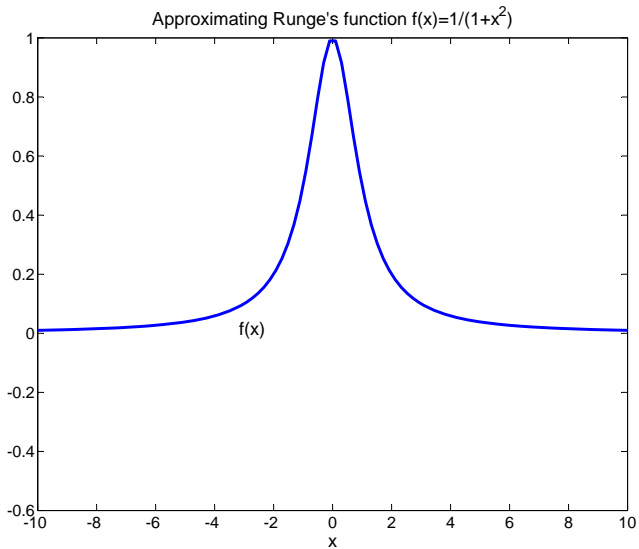
Figure 3: Chebyshev Polynomial Basis Functions



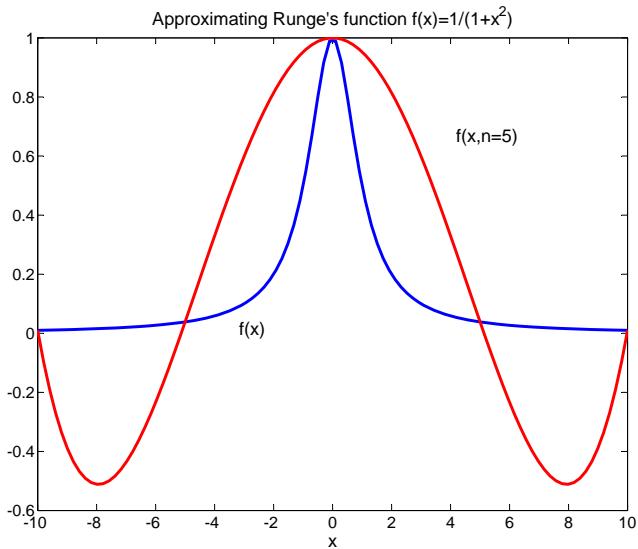
## Choice of nodes

- Equidistant nodes BAD IDEA
- Example:  $f(x) = \frac{1}{1+x^2}$

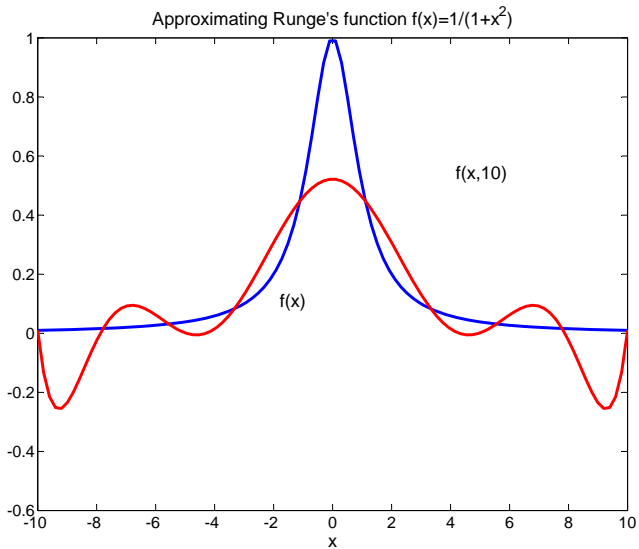
# Equidistant nodes



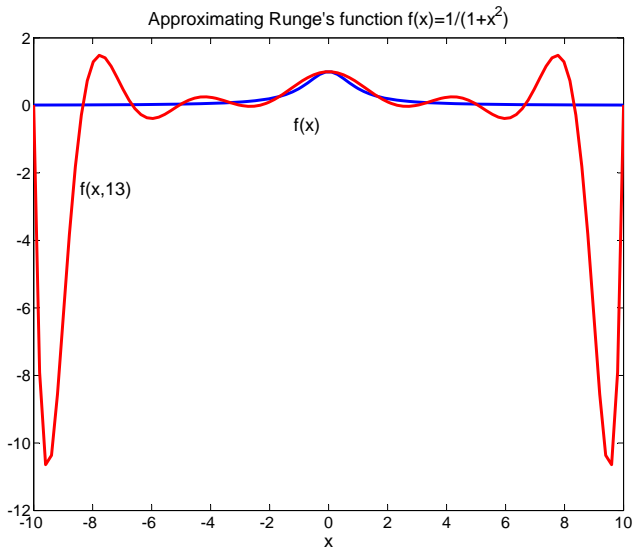
# Equidistant nodes



# Equidistant nodes



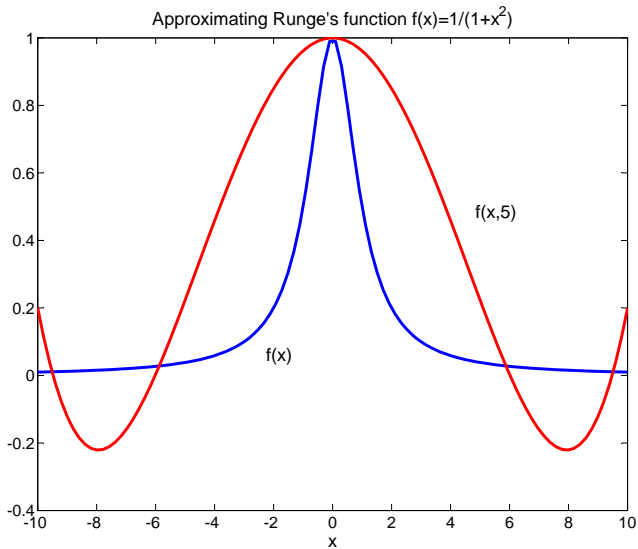
# Equidistant nodes



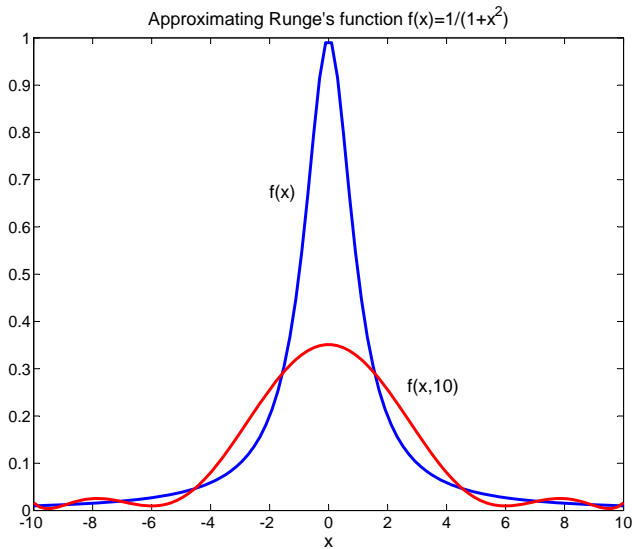
## Optimal choice of interpolation nodes

- Chebyshev nodes (roots of  $n$ -th degree Chebyshev polynomial) nearly optimal if certain smoothness conditions satisfied
- Rivlin's theorem:
  - If  $f : [a, b] \rightarrow R$  is  $C^k$  for some  $k \geq 1$  and  $I_n$  is interpolant at zeros of Chebyshev polynomials  $\phi_n(x)$ , then:
    - $\|f - I_n\|_\infty \leq \left(\frac{2}{\pi} \log(n+1) + 1\right) \frac{(n-k)!}{n!} \left(\frac{\pi}{2}\right)^k \left(\frac{b-a}{2}\right)^k \|f^{(k)}\|$
    - E.g.,  $k = 1$ ,  $bound = \frac{(\frac{2}{\pi} \log(n+1) + 1)}{n} \frac{\pi}{4} \|f^{(1)}\| (b-a)$

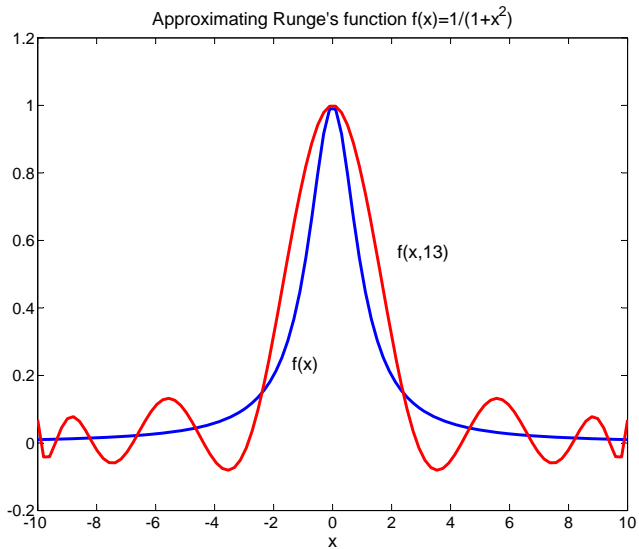
# Chebyshev nodes



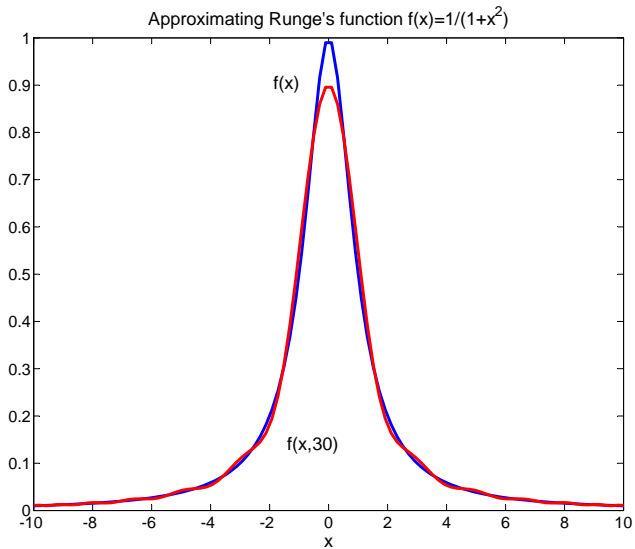
# Chebyshev nodes



# Chebyshev nodes



# Chebyshev nodes



## Solving for unknown coefficients

- Choose  $n$ : order of polynomials
- Choose  $m \geq$ : number of nodes,  $x = x_1, \dots, x_m$ ,
- Evaluate  $F = f(x)$
- Choose  $c = c_1, \dots, c_n$  to min:  $(\Phi c - F)'(\Phi c - F)$

$$\Phi = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) & \dots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \phi_3(x_2) & \dots & \phi_n(x_2) \\ \phi_1(x_3) & \phi_2(x_3) & \phi_3(x_3) & \dots & \phi_n(x_3) \\ \dots & \dots & \dots & \dots & \dots \\ \phi_1(x_m) & \phi_2(x_m) & \phi_3(x_m) & \dots & \phi_n(x_m) \end{pmatrix}$$

- $\Phi$  very well-conditioned: (use `c=Phi\F` in Matlab)

## Solving for unknown coefficients

Interpolate runge's function on  $[-10,10]$

```
n=[7]; /* order of polynomial*/
m=[15]; /*number of nodes*/
a=-10;b=10 /* range of approximation*/

X=chebnode(m,a,b); /*roots of m-th order chebyshev polynomial: nodes*/

F=1./(1+x.^2); /* evaluate function*/

Phi=chebbas(n,a,b,x); /* evaluate matrix of basis functions at x*/

c=Phi\F; /* unknown coefficients*/
xnode=(-10:.01:10)'; /* denser grid of points*/
Phinode=chebbas(n,a,b,xnode); /* basis functions at xnode*/
plot(xnode,[Phinode*c,1./(1+xnode.^2)]); /* plot*/
```

## Extension to $>1$ dimension

- Straightforward with tensor products, but curse of dimensionality
- E.g., approximate  $f(x, y)$  on  $[a_x, b_x] \times [a_x, b_x]$
- Generate univariate nodes  $x = x_1, \dots, x_{n_x}, y = y_1, \dots, y_{n_y}$
- Compute univariate basis functions  $\phi_i^x, \phi_i^y$
- Approximant:  $I = \sum_{i_x=1}^{n_x} \sum_{i_y=1}^{n_y} c_{i_x i_y} \phi_{i_x}^x(x) \phi_{i_y}^y(y)$
- Compact notation:  $I = \Phi c$ , where  $\Phi = \Phi_k \otimes \Phi_{k-1} \otimes \dots \otimes \Phi_1$

## Example in Compecon

Interpolate  $f(x, y) = \frac{1}{(1+x^2y^2)}$

```
n=[7,7]; /* nodes in each dimension */
```

```
a=[-10,-10]; b=[10,10]; /* bounds for each dimension*/
```

```
fspace=fundefn('cheb',n,a,b) /* define func. approx. space*/
```

```
xnode=funnode(fspace); /*default nodes (i.e., m=n, use chebnode otherwise)*/
```

```
xnode=gridmake(xnode); /* Cartesian product of univariate nodes */
```

```
Phi=funbas(fspace,xnode,[0,0]); /* basis function at Cartesian product:
```

```
last argument specifies order of differentiation in each dimension*/
```

```
F=1./(1+xnode(:,1).^2.*xnode(:,2).^2); /*eval. function at nodes */
```

```
c=Phi\F; /*coefficients*/
```

# Splines

- Alternative function approximation space
- Divide  $D$  along each dimension into subintervals
- Approximate  $f$  on which subinterval with splines (low-order polynomials)
- Require value matching and smooth pasting in addition to earlier conditions to pin down  $c$
- Ensure errors in one region to not spill over to over regions
- See `splibas`, `splinode`, `fundefn(spli',...)`, `fundefn('lin',...)` for linear and cubic splines
- Same implementation as above

# Solving functional equations

- Straightforward using tools above
- E.g., Solve for  $f() : g(f(.)) = 0$
- Approximate  $f$  with  $f_n(., c)$
- Solve for  $c : g(f_n(., c)) = 0$  at Chebyshev nodes in state-space