MATLAB-Kurs

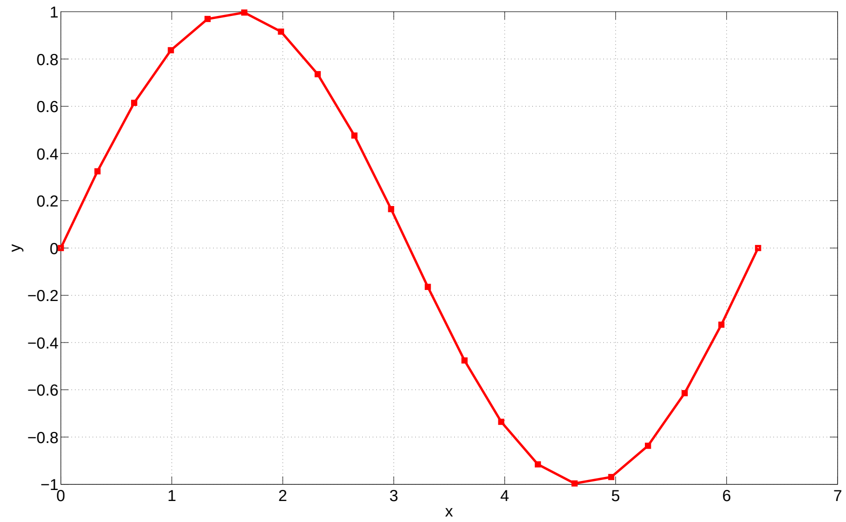
Übersicht

- Einführung in MATLAB
- Quadratur mit MATLAB
- Eigenwertberechnung in MATLAB

1. Einführung in MATLAB

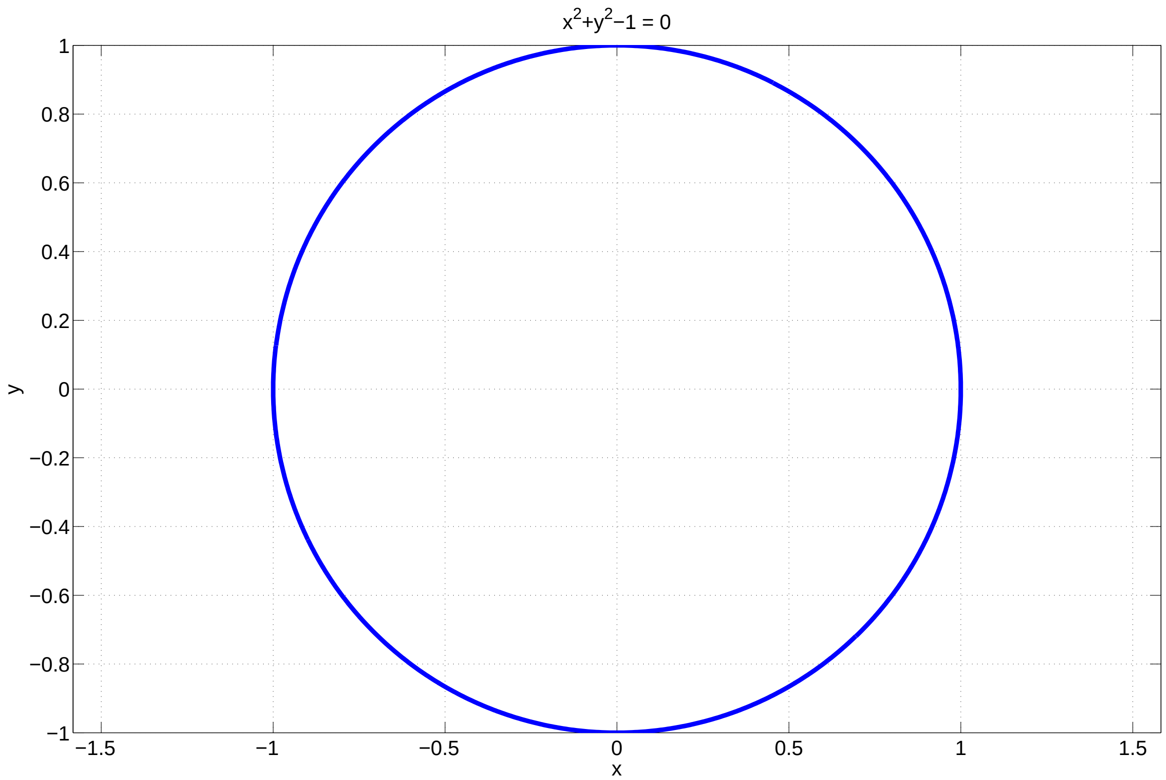
Plotten von Funktionen mit einer Veränderlichen

```
>> x = linspace(0,2*pi,20);
>> y = sin(x);
>> plot(x,y,'rs-','LineWidth',3)
>> grid on
```



Plotten von implizit definierten Funktionen in 2D

```
>> ezplot('x^2+y^2-1',[-1,1]);
>> axis equal;
```



3D-Grafik

- Zum Auswerten einer Funktion $f(x, y)$ (zwei unabhängige Variablen) zunächst ein 2D-Gitter erzeugen:

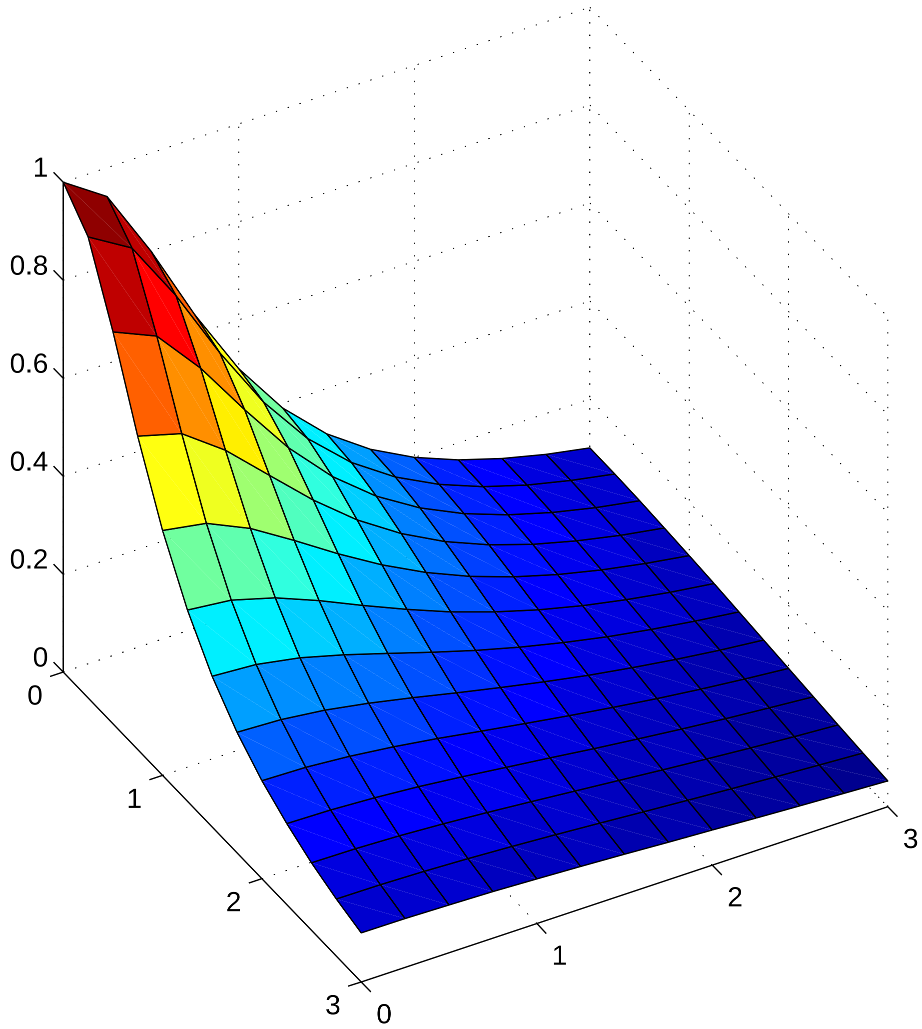
```
>> x = 0:3;
>> y = 0:3;
>> [X,Y] = meshgrid(x,y);
```

$$X = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \end{pmatrix}$$

- Dann Funktion auswerten:

```
>> Z = 1./(1+X.^2+Y.^2)
```

$$Z = \begin{pmatrix} 1 & 1/2 & 1/5 & 1/10 \\ 1 & 1/3 & 1/6 & 1/11 \\ 1 & 1/6 & 1/9 & 1/14 \\ 1 & 1/11 & 1/14 & 1/19 \end{pmatrix}$$



- Zuletzt plotten:

```
>> surf(X,Y,Z)
```

oder

```
>> mesh(X,Y,Z)
```

Logarithmische Plots

- Beide Achsen logarithmisch:

```
>> h = 10.^(-1:-1:-17)
>> y = fun(h)
>> loglog(x,y)
```

- x-Achse logarithmisch:

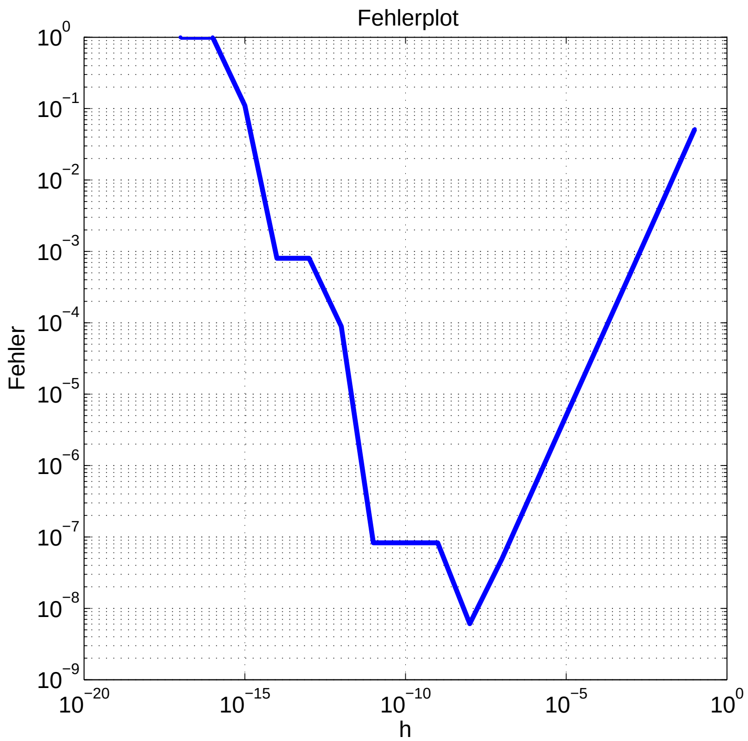
```
semilogx
```

- y-Achse logarithmisch:

```
semilogy
```

Logarithmische Plots

- Beispiel: $\left|\exp(0) - \frac{\exp(h)-\exp(0)}{h}\right|$



Formatierung und Beschriftung der Grafiken

- Titel einfügen:

```
>> title('Verlauf der Funktion y=x^2')
```

- Beschriftung der x- (bzw. y-) Achse:

```
>> xlabel('Zeit in [s]')
```

- Text an spezifizierten Koordinaten einfügen:

```
>> text(1,1,'Wertebereich: [0,1]')
```

- Text interaktiv mit der Maus einfügen (2D):

```
>> gtext('Funktion f')
```

- Legende erstellen:

```
>> legend('Funktion f','Funktion g')
```

- Gitter ein/ausblenden:

```
>> grid
```

- Farbwerteskala einblenden:

```
>> colorbar
```

- Gleiches Achsenverhältnis einstellen:

```
>> axis square
```

- Weitere Plots in die bestehende Grafik einfügen:

```
>> hold on
```

(Danach mit *hold off* wieder deaktivieren)

- Grafik als eps-Datei ausdrucken:

```
>> print -depsc meinbild.eps
```

Mathematische Funktionen in Matlab – Übersicht

- Eingabe von Funktionen mit `inline`
- Übergabe von mathematischen Funktionen an Funktionen
- Nullstellensuche für Funktionen mit einer Unbekannten
- Minimieren von Funktionen mit einer Unbekannten
- Minimieren von Funktionen mit mehreren Unbekannten

Eingabe von Funktionen mit `inline`

- Alternativ zur Funktionen-Definition mit m-Files kann auch die `inline`-Funktion verwendet werden.

- Beispiel: $f(x) = \frac{1}{1+x^2} + \frac{1}{(x-5)^2+2} - 0.2$:

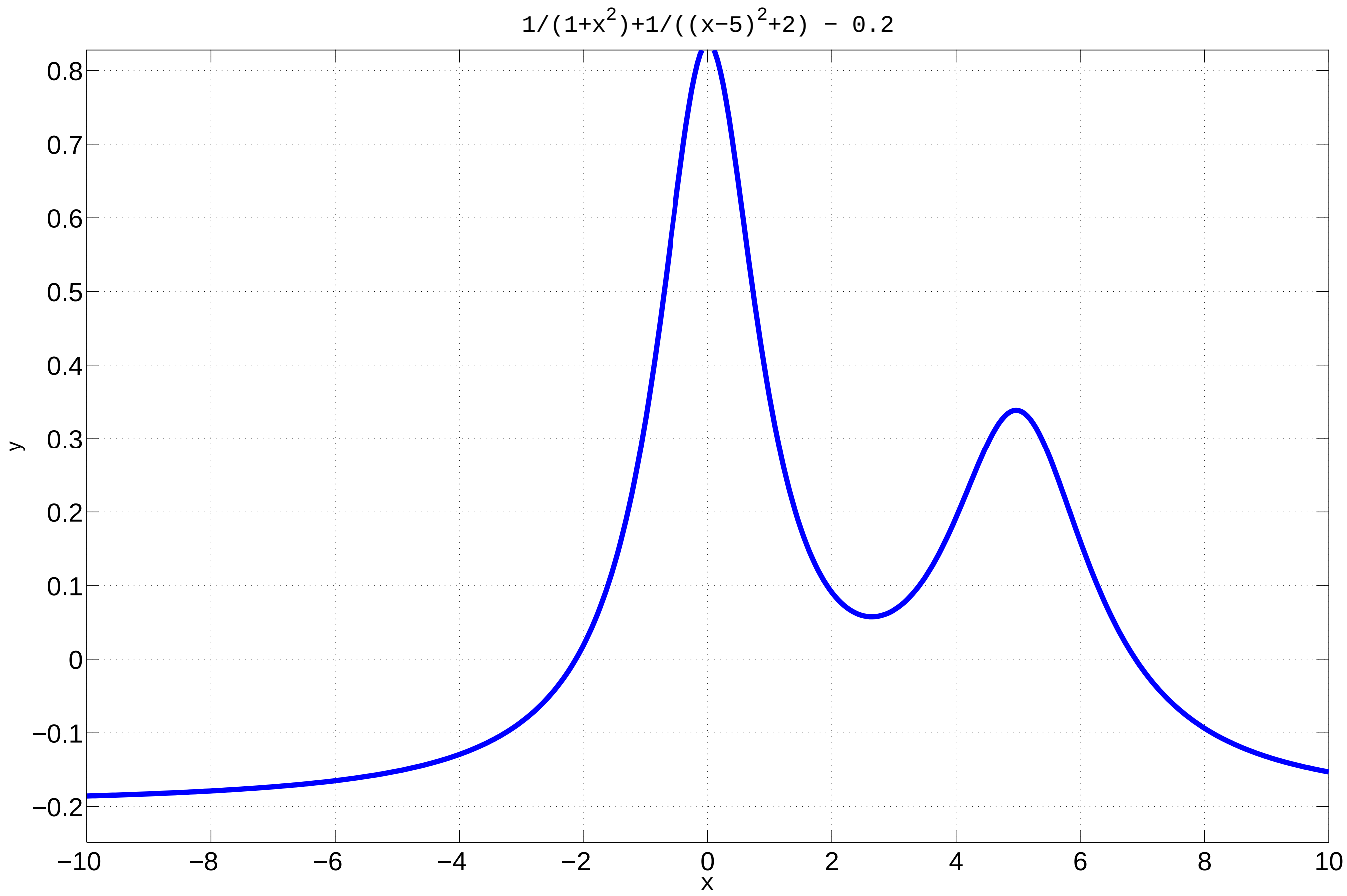
```
>> f = inline('1./(1+x.^2)+1./((x-5).^2+2) - 0.2');
```

- Die Variable x wird automatisch erkannt.

- Man kann die Funktion f dann wie gewohnt (d.h. wie ein m-File) aufrufen, z.B.:

```
>> f(0)
ans = 0.8370
```

- Plotten ist ebenfalls wie gewohnt möglich, z.B. mit `ezplot(f,[-10 10])`:



Inline-Funktionen mit mehreren Variablen

- Namen der Variablen mit dem Ausdruck angeben

- Beispiel:

```
>> f = inline('cos(x).^2+sin(y).^2', 'x', 'y');
```

- **Achtung:** Die Variablennamen in Anführungsstrichen angeben!

- Mit dem Befehl `symvar` kann man eine inline-Funktion nach Variablennamen durchsuchen lassen. Beispiel:

```
>> symvar('cos(pi*x - alpha)')
ans =
    'alpha'
    'x'
```

Übergabe von Funktionen an andere Funktionen

- **Übergabe von m-File functions:**
  mit sog. function handles:
  - Erzeugen des "handle" mit dem @-Symbol, z.B. `fun = @f_stck`, wenn im Verzeichnis ein m-File mit Namen `f_stck` existiert.
  - Übergabe des handle dann als Parameter, z.B.:
    `>> bisection(fun, 0, 1, 1e-5, 10);`
- **Übergabe von inline-functions:**
  Direkt die Inline-Variable übergeben (dies ist bereits ein handle).
- **Auswerten von übergebenen Funktionen**
  - Mit dem Befehl `feval`. Beispiel:
    `>> feval(fun, 2)` oder `>> feval(@f_stck, 2)`
    entspricht
    `>> f_stck(2)`
  - Achtung: Übergebene Funktionen (handles) können nicht direkt ausgewertet werden, d.h. man **muss** `feval` verwenden.

Nullstellensuche mit `fzero`

- Für Funktionen mit einer Unbekannten
- Einfache Form: Angabe der Funktion und Wert nahe Nullstelle bzw. Intervall, das Nullstelle enthält.

```
>> f = inline('1./(1+x.^2)+1./((x-5).^2+2) - 0.2');
>> x = fzero(f, -2)
x = -2.1268
>> x = fzero(f, [-10 4])
x = -2.1268
```

- **Einschränkungen:**
  - Falls die Funktion keinen Vorzeichenwechsel auf der reellen Achse besitzt, wird `NaN` zurückgegeben.
  - Falls die Funktion eine Unstetigkeitsstelle mit Vorzeichenwechsel besitzt, wird dies numerisch als Nullstelle interpretiert
    (z.B. bei $f(x) = 1/x$ oder $f(x) = \tan(x)$).

- **Erweiterter Aufruf von `fzero`:**
  - Spezifizieren von Optionen mit `optimset`
    (z.B. erweiterte Anzeige, Toleranz, Anzahl Iterationen angeben)
  - Siehe auch `help optimset`
  - Beispiel:

```
>> options = optimset('Display','iter');
>> f = inline('1./(1+x.^2)+1./((x-5).^2+2) - 0.2');
>> x = fzero(f, [-3, -2], options);
 Func-count        x              f(x)           Procedure
     1               -3     -0.0848485            initial
     2               -2      0.0196078            initial
     3         -2.18771    -0.00853697            interpolation
     4         -2.13078    -0.00057904            interpolation
     5         -2.12675    1.43791e-06            interpolation
     6         -2.12676   -3.02089e-09            interpolation
     7         -2.12676   -1.57374e-14            interpolation
     8         -2.12676              0            interpolation
Zero found in the interval: [-3, -2].
```

Minimieren von Funktionen mit einer Unbekannten mit `fminbnd`

- Ermittlung des **lokalen** Minimums

- Angabe des Intervalls ist erforderlich

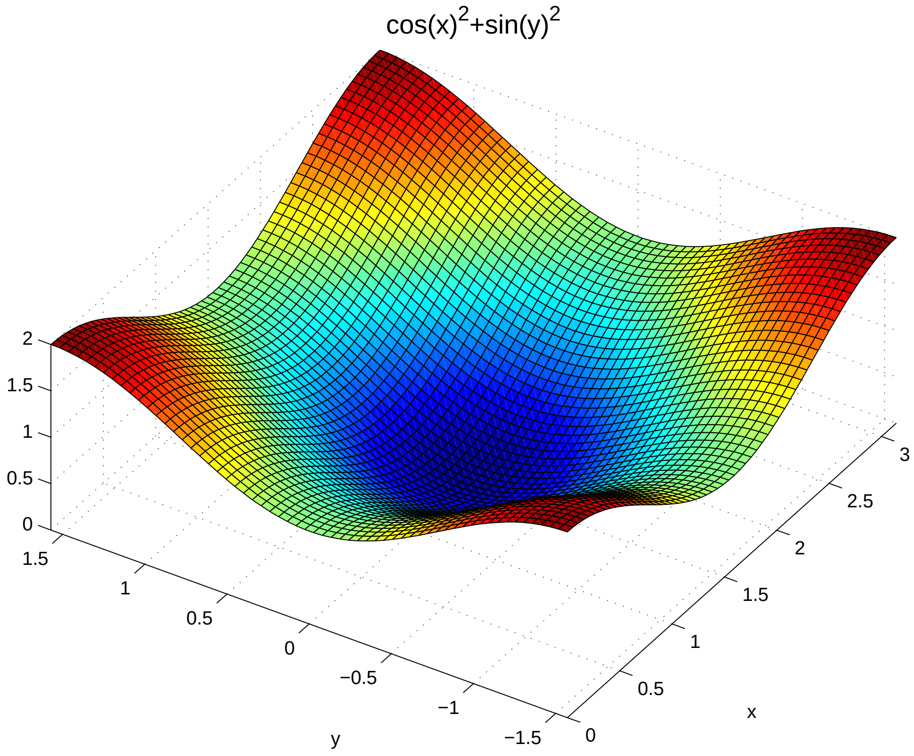
- Beispiel:

```
>> f = inline('1./(1+x.^2)+1./((x-5).^2+2) - 0.2');
>> x = fminbnd(f,0,6)
x = 2.6451
>> x = fminbnd(f,-10,6)
x = -10.0000
```

- Wie bei `fzero` ist die Angabe von Parametern durch `optimset` möglich.

Minimierung bei mehreren Unbekannten: `fminsearch`

- **Achtung:** Die Funktion muss die einzelnen Variablen in Form einer Matrix aktzeptieren! (`inline` daher nicht geeignet)
- **Beispiel:** Bestimmen eines lokalen Minimums von $f(x,y) = \cos^2(x) + \sin^2(y)$



- Definition der Funktion als m-File:

```
function f = f_2var(v)
  % v ist in diesem Fall ein Zeilenvektor
  x = v(1);
  y = v(2);
  f = cos(x)^2 + sin(y)^2;
```

- Beachte Unterschied im Aufruf, z.B. Auswertung :

```
>> f_2var([0 pi/2])
ans =
     2
```

- Minimierung für das Gebiet $[0, \pi] \times [-\pi/2, \pi/2]$, Startvektor $s = (0, 0)$:

```
>> v = fminsearch(@f_2var, [0 0])
v =
    1.5708     0.0000
```

- Wie bei `fzero` ist die Angabe von Parametern durch `optimset` möglich.

Schwach besetzte Matrizen in Matlab - Übersicht

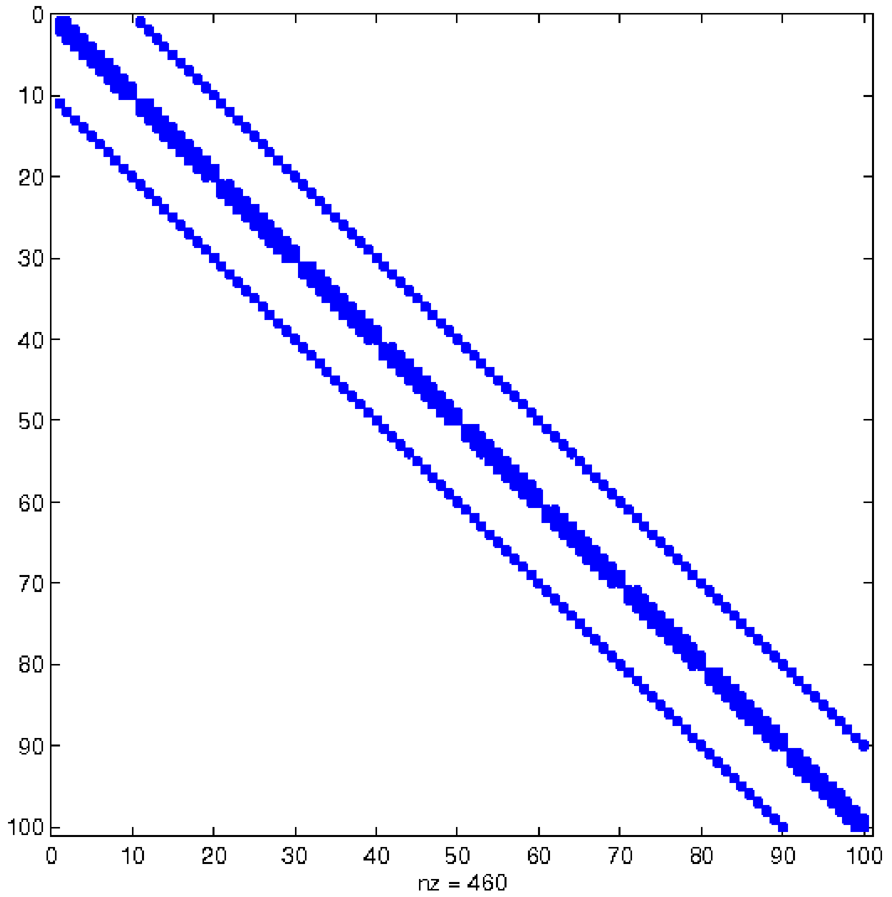
- Motivation
- Speichermodell von Sparse-Matrizen
- Erzeugen und Arbeiten mit Sparse-Matrizen
- Vergleich mit voll besetzten Matrizen
- Erhalten von schwach besetzten Strukturen

Motivation

Für viele Probleme (z.B. bei der Diskretisierung partieller Differentialgleichungen) erhält man Matrizen, die viele Nullen enthalten.

$$A = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix}$$

Betrachten der Matrixstruktur mit `spy`



Ziele für den Umgang mit schwach besetzten Matrizen

- **Speicherplatz sparen:**
  nur Nicht-Null-Einträge werden abgespeichert

- **Effizientes Rechnen:**
  Explizite Operation mit Null-Einträgen vermeiden

Durch diese Effizienzsteigerungen können Probleme berechnet werden, die ansonsten aufgrund ihrer Größe nicht verarbeitet werden könnten.

Speichermodell von Sparse-Matrizen

- Nur von Null verschiedene Einträge werden abgespeichert
- Der Ort des Eintrags wird durch den Zeilen- und Spaltenindex gekennzeichnet
- Speicherplatzbedarf ist ungefähr gleich der Summe von
  - 4 Bytes, um die Anzahl der Einträge zu speichern
  - pro Eintrag 8 Bytes für Spalten- und Zeilenindex
  - pro Eintrag 8 Bytes für den Zahlenwert

Beispiele für Sparse-Matrizen

- Konvertierung von B nach schwach bzw. voll besetzt:
- `full(B)`
- `sparse(B)`

$$B = \begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

```
>> sparse(B)

ans =

   (1,1)        2
   (2,1)       -1
   (1,2)       -1
   (2,2)        2
    ...
```

Befehle zum Erzeugen von Sparse-Matrizen

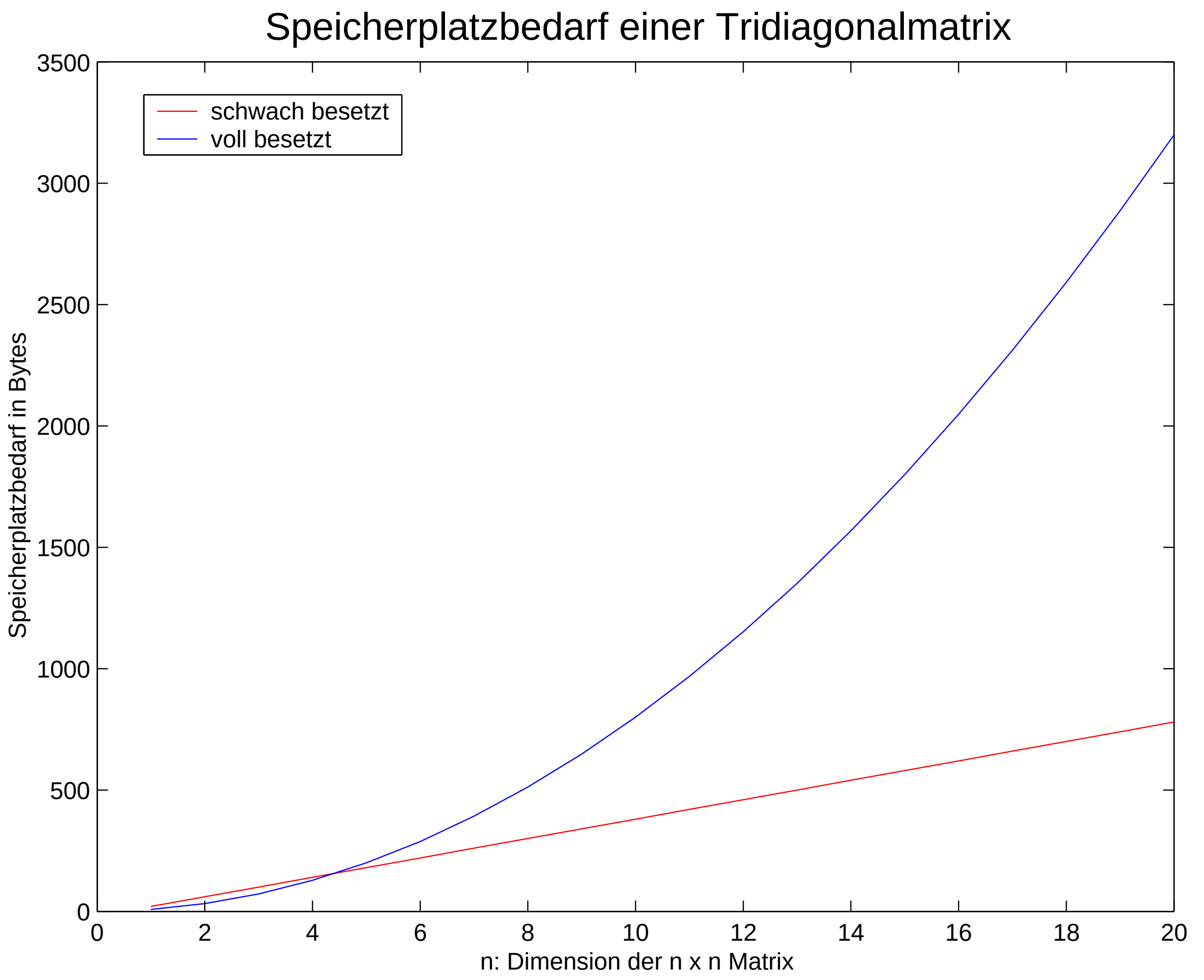
Funktion	Beschreibung
`speye`	Schwach besetzte Einheitsmatrix
`spones`	Einträge durch Einsen ersetzen
`spdiags`	Erzeugen von Bandmatrizen
`sprand`	Einträge durch Zufallszahlen ersetzen

Erzeugen von Bandmatrizen

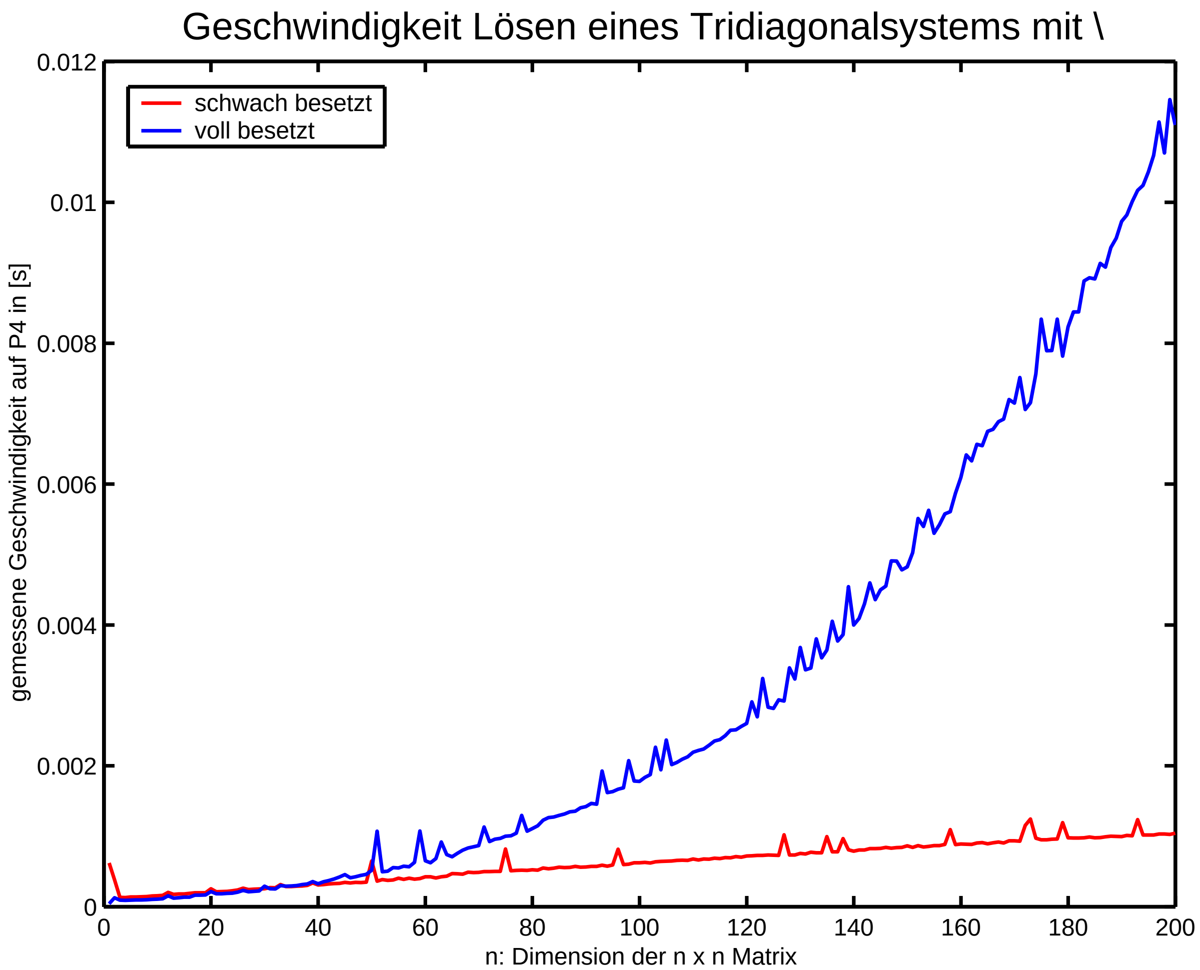
mit spdiags:

```
>> n = 5; e = ones(n,1); d = 4*e;
>> A = spdiags([-e, d, -e], [-1,0,1], n, n);
>> full(A)
ans =
     4    -1     0     0     0
    -1     4    -1     0     0
     0    -1     4    -1     0
     0     0    -1     4    -1
     0     0     0    -1     4
```

Vergleich Speicherplatzbedarf



Vergleich Rechengeschwindigkeit



Erhalten schwach besetzter Strukturen

Beispiel: Invertieren einer Matrix

- in der Regel ist die Inverse einer schwach besetzten Matrix nicht mehr schwach besetzt.
- Daher: Oft besser, Inverse nicht explizit zu bestimmen. Stattdessen kann man oft ein lineares Gleichungssystem (z.B. mit dem \ Operator) lösen.

Beispiel: SOR-Verfahren (1)

Falsch: Mit Berechnung der Inversen:

```
function [u,m]=solveSORWrong(A, f, u_s, w, tol, m_max);

  D = diag(diag(A));  L = tril(A,-1);
  inv_C_SOR = inv(1/w*(D+w*L));      % C_SOR = 1/w*(D+wL)

  u = u_s;     % Setze u gleich dem Startvektor
  r = f - A*u  % Berechne das erste Residuum;
  m=0;

  while ((norm(r)/norm(f)>tol) & (m<m_max))
    % Iterationsschritt:
    u = u + inv_C_SOR*r;
    r = f - A*u;
    m = m + 1;
  end
```

Beispiel: SOR-Verfahren (2)

Richtig: Vermeiden der Inversen durch Verwenden des \ Operators:

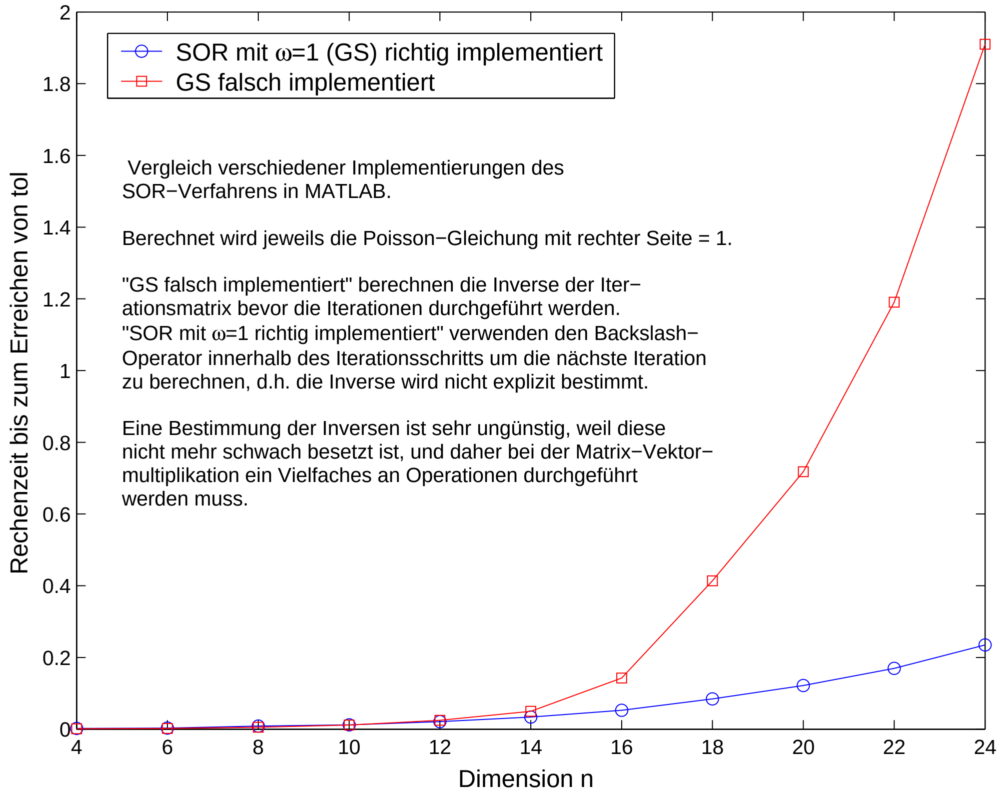
```
function [u,m]=solveSOR(A, f, u_s, w, tol, m_max);

  D = diag(diag(A));  L = tril(A,-1);
  C_SOR = 1/w*(D+w*L);                    % C_SOR = 1/w*(D+wL)

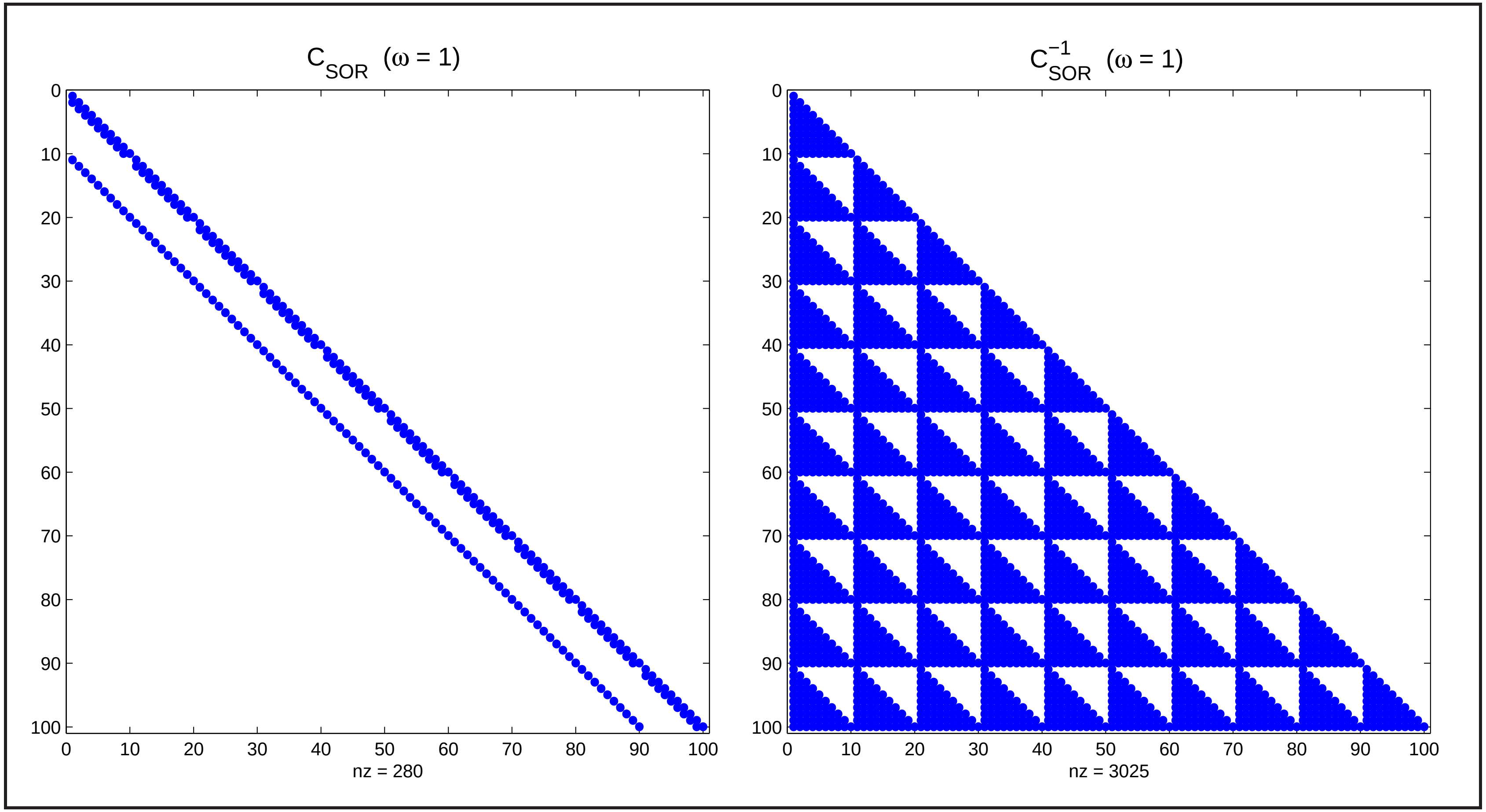
  u = u_s;     % Setze u gleich dem Startvektor
  r = f - A*u  % Berechne das erste Residuum;
  m=0;

  while ((norm(r)/norm(f)>tol) & (m<m_max))
    % Iterationsschritt:
    u = u + C_SOR\r;
    r = f - A*u;
    m = m + 1;
  end
```

SOR - Geschwindigkeitsvergleich



Matrixstruktur $C_{SOR} = D + L$



2. Numerische Quadratur mit MATLAB

Wiederholung

- Numerische Quadratur: Approximative Bestimmung von Integralwerten$I(f)$ durch eine Quadraturformel:

$$Q_a^b(f) := \sum_{i=0}^{n} w_i f(x_i)$$

Stützstellen: $x_i \in [a, b]$, Gewichte der Quadraturformel: w_i

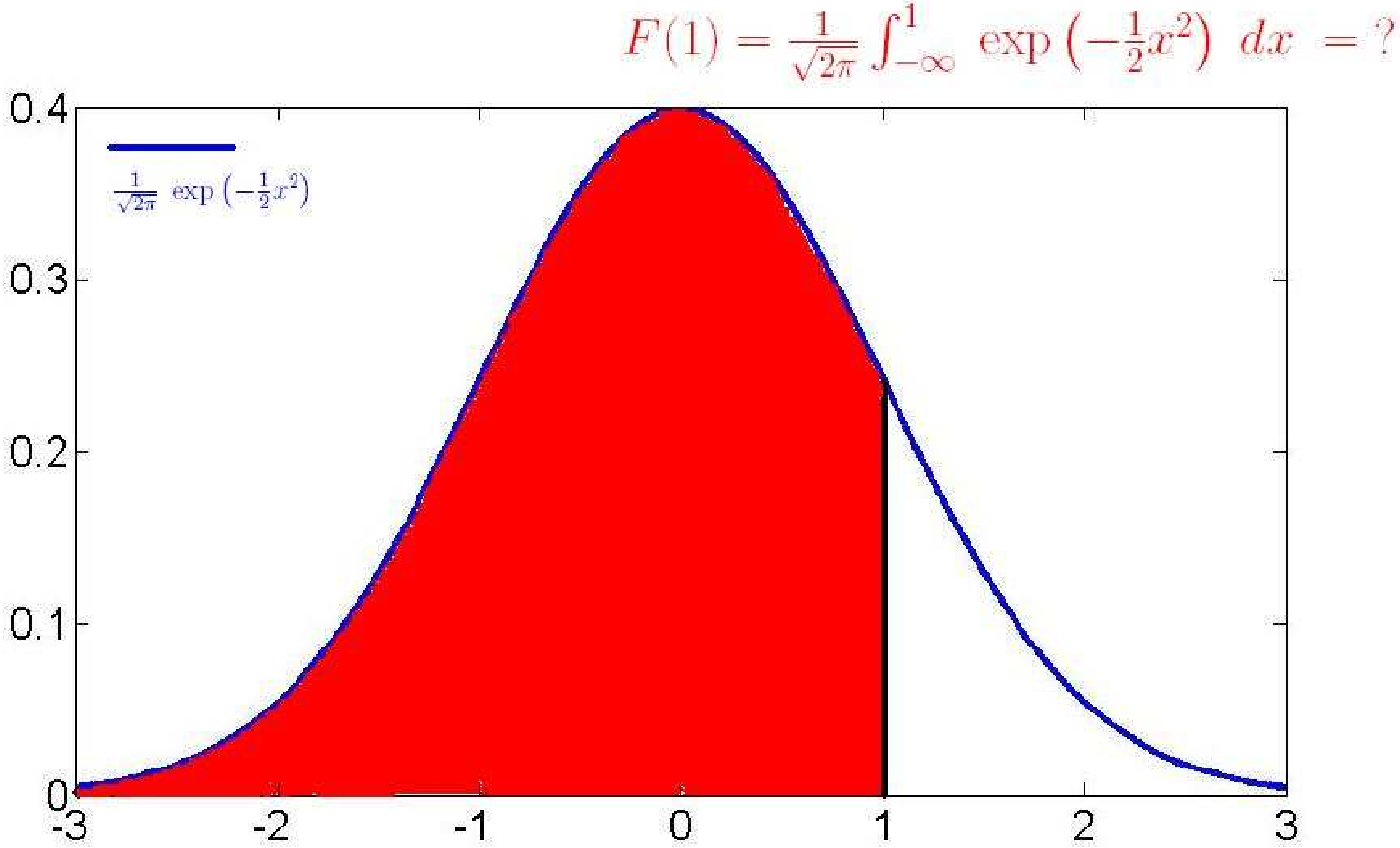
- Beispiele:
(1) Berechnung von Funktionswerten der Gammafunktion:

$$\Gamma(x) = \int_0^\infty t^{x-1} \exp(-t)\, dt$$

(2) Berechnung von Funktionswerten der Verteilungsfunktion der

Normalverteilung $N(0,1)$:

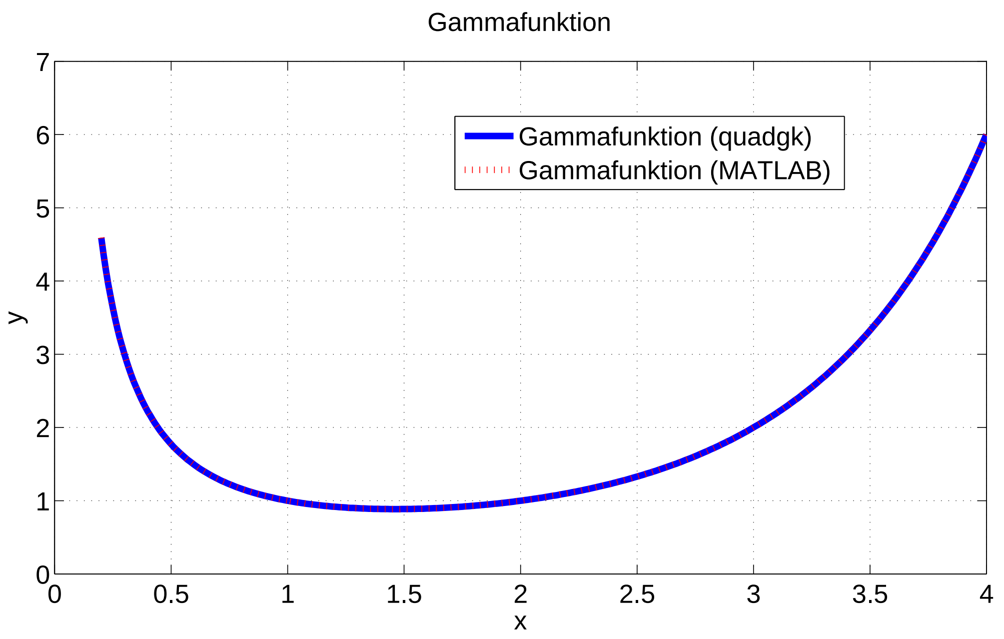
$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{1}{2}t^2\right) \, dt$$



Überblick

- quad (Adaptive Simpson-Quadratur, siehe VL)
- quadl (Adaptive Gauß-Lobatto Quadratur)
- quadgk (Berechnung uneigentlicher Integrale)
- dblquad (Berechnung von Doppel-Integralen)
- triplequad (Berechnung von Dreifach-Integralen)

- Beispiel: Gammafunktion berechnet mit quadgk:
  $\Gamma(x) = \int_0^\infty t^{x-1} \exp(-t)\, dt$



quadl

- quadl arbeitet ähnlich wie quad

- Das ursprüngliche Intervall wird in kleinere Teilintervalle zerteilt

- Auf jedem Intervall wird der Fehler durch die Differenz zweier Approximationen I_1 und I_2 geschätzt (I_1 hat eine höhere Ordnung als I_2)

- Ein Teilintervall wird nicht mehr weiter verfeinert, falls gilt: $|I_2 - I_1| \leq TOL * |I_1|$. TOL ist eine vorgegebene Toleranz. $TOL = 10^{-6}$ ist die Standardeinstellung bei quadl.

- Gauß-Lobatto Quadraturformel:
$I_2(f) = \frac{1}{6}[f(-1) + f(1)] + \frac{5}{6}\left[f(-\frac{1}{\sqrt{5}}) + f(\frac{1}{\sqrt{5}})\right]$ (Exaktheitsgrad 5)

- Kronrod-Erweiterung:
$I_1(f) = \frac{11}{210}[f(-1) + f(1)] + \frac{72}{245}\left[f\left(-\sqrt{\frac{2}{3}}\right) + f\left(\sqrt{\frac{2}{3}}\right)\right] + \left[f\left(-\frac{1}{\sqrt{5}}\right) + f\left(\frac{1}{\sqrt{5}}\right)\right] + \frac{16}{35}f(0)$ (Exaktheitsgrad 9)

quadl

- Numerische Approximation von $\int_{-1}^{1} f(x)\,dx$ mittels quadl

- $f(x) = \begin{cases} 1000\exp(-\frac{1}{x(0.01-x)}) & \text{für } 0 < x < 0.01 \\ 0 & \text{sonst.} \end{cases}$

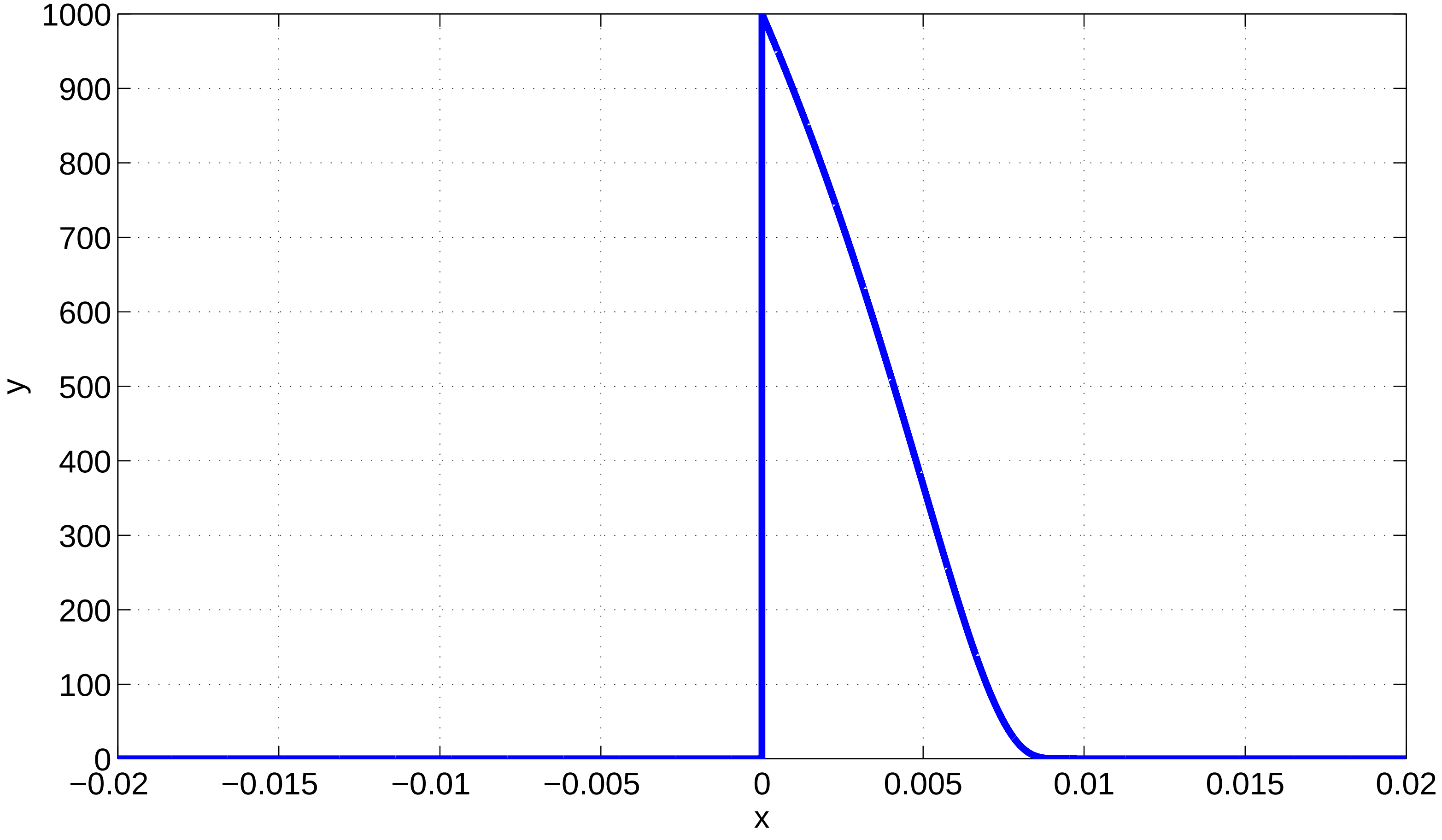
- ```
  >> quadl(@f,-1,1)
  >> ans = 0
  ```

  obwohl $I(f) > 0$

- Erklärung: Alle Stützstellen von $I_1(f)$ und von $I_2(f)$ liegen außerhalb des Trägers von f, d.h. $I_1(f) = 0$ und $I_2(f) = 0$.

- Abhilfe:

  ```
  >> quadl(@f,-1,0.001)+quadl(@f,0.001,1)
  >> ans = 4.0365
  ```
  ```


1000
900
800
700
600
500
400
300
200
100
0
y
−0.02
−0.015
−0.01
−0.005
0
0.005
0.01
0.015
0.02
x

3. Eigenwertberechnung mit MATLAB

Problemstellung

- Problemstellung:
  Bestimmung der Eigenwerte und Eigenvektoren einer quadratischen Matrix $A \in \mathbb{R}^{N \times N}$

- $Ax = \lambda x, \quad x \neq 0$

- λ Eigenwert, x Eigenvektor

MATLAB-Funktionen

- eig (liefert alle Vektoren und Eigenwerte)
- eigs(liefert standardmäßig die sechs größten Eigenwerte und die dazugehörigen Eigenvektoren)