Programmieraufgabe I (9 + 8 + 4 + 4 = 25 Punkte)

Abgabe in der Woche vom 5. bis 9. November

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Die grundlegenden Operationen der linearen Algebra (d.h. Summen, Skalarprodukte, Norm, Matrix-Vektor-Produkt, etc.) sind sind für numpy-arrays bereits implementiert und dürfen hier verwendet werden.

Teilaufgabe a)

Implementieren Sie eine Funktion, die als Input ein Matrix A (als numpy-array) erhält und die reduzierte QR-Zerlegung ($\hat{Q}$, $\hat{R}$ jeweils als numpy-arrays) zurückgibt. Als Subroutine empfiehlt es sich, die Anwendung der Householder-Transformation separat zu implementieren.

In [64]:
```
def myqr(A):

    ## Diese Funktion soll die reduzierte QR-Zerlegung von A berechnen und Q
,R (in der Vorlesung $\hat{Q}, \hat{R}$) zurückgeben...

    return Q,R

def householder(arg, w):

    ## Diese Funktion soll eine Householder-Transformation $I - 2ww^T$ auf d
as Argument arg (Matrix oder Vektor) anwenden.

    return result
```

Testen Sie Ihre eigene Implementierung, indem Sie eine zufällige 5×4-Matrix erzeugen und deren mit Ihrer Implementierung generierte QR-Zerlegung mit der in numpy bereits implementierten QR-Zerlegung vergleichen.

In [65]:
```
## Hinweis: Verwenden Sie die Generierung zufälliger Matrizen aus numpy...
```

Teilaufgabe b)

Implementieren Sie die QR-basierte Lösungsstrategie für das Least-Squares Problem $\min_x ||Ax - b||$: Die Funktion soll als Input die Matrix A und den Vektor b (beide als numpy-array) erhalten und die Lösung $x_{\min}$ zurückgegebn.

Die Lösung eines Gleichungsystems mit oberer Dreiecksstruktur sollen Sie als eigene Routine programmieren. (Der lineare Löser aus numpy soll hier nicht verwendet werden.)

In [4]:
```
def myuppertriangularsolver(A,b):

    ## Löst die Gleichung Ax = b, wobei A eine obere Dreiecksmatrix mit stri
kt positiven Diagonalementen ist. ##

    return x

def myleastsquares(A,b):

    ## Löst das least squares problem $min_x \lVert Ax-b \rVert_2$.

    return xmin
```

Testen Sie Ihre Implementierung an Aufgabe 1b) von Übungsblatt Nr. 2:

In [66]:
```
b = np.array([2, 3, 3, -2])
A = np.array([[1, -2, 4], [1, -1, 1], [1,1,1], [1,2,4]])
xtrue=np.array([4, -4./5, -1])

## Test: ...
```

Teilaufgabe c)

Es sei das Interval $I = (0, 1)$ gegeben. Die folgende Funktion gibt einen Vektor von $x-$Werten aus I sowie die geringfügig zufällig gestörten zugehörigen $y-$Werte $y \approx f(x)$ zurück.

In [58]:
```
def noisyfunction(fun, numberofpoints, noisemax):
    xvals = np.random.rand(numberofpoints)
    yvals = np.zeros(numberofpoints)
    for k in range(numberofpoints):
        yvals[k] = fun(xvals[k]) + 2*noisemax*(np.random.rand()-0.5)
    return xvals, yvals
```

Es sei eine Funktion $F : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ gegeben, z.B. $F_{\mathrm{Pol}}(x, l) := x^l$ oder $F_{\sin}(x, l) = \sin(\pi l x)$, derart, dass durch $f_l(x) := F(x, l)$ eine "sinnvolle" Familie von Basisfunktionen definiert wird.

Für gegebene Datenpunkte $(x_1, y_1), \ldots, (x_m, y_m)$ und $n \in \mathbb{N}$ wollen wir einen Koeffizientenvektor $c = (c_0, \ldots, c_n)$ bestimmen, der das Least-Squares Problem $\min_{c\in\mathbb{R}^n} \sum_{k=1}^{m} |\sum_{l=1}^{n} c_l f_l(x_k) - y_k|^2$ löst.
Schreiben Sie dazu die Funktion "myregression" mit folgenden Inputs:

- xvals, yvals: Arrays mit den gegebenen Wertepaaren
- funfamily: Funktion F für die Erzeugung der Basisfunktionen (siehe oben)
- degree: entspricht n, der Anzahl der verwendeten Basisfunktionen
- xtoevaluate: array mit $x-$Werten, an denen die gefundene Regressions-Lösung ausgewertet werden soll

Ausgegeben werden soll der Koeffizientenvektor coeffvec sowie die Auswertung y der Regressions-Lösung an den in xtoevaluate gegebenen x-Werten.

In [68]:
```
def myregression(xvals, yvals, funfamily, degree, xtoevaluate):

    ## ... ##

    return y, coeffvec

## Implementierung der Basis-Familien $F_Pol$ bzw. $F_sin$ von oben:
def polynomialfamily(x,l):
    return x**l

def sinefamily(x,l):
    return np.sin(np.pi*x*(l+1))
```

Teilaufgabe d)

Testen Sie Ihre Implementierung für die Lösung des Least-Squares Problems mit den unten erzeugten Wertepaaren:

- Plotten Sie dazu jeweils die verrauschten Wertepaare sowie die Regressionsfunktion.
- Variieren Sie die Anzahl der Basisfunktionen bzw. die Wahl der Familie von Basisfunktionen.

Erklären Sie, weshalb die Lösung des Least-Squares Problems für die Approximation von $x - \frac{1}{2}$ mittels Sinus-Schwingungen sehr viel instabiler zu sein scheint als die Approximation des Sinus mit Polynomen.

In [71]:
```
def fun_1(x):
    return np.sin(2*np.pi*x)
xvals_1,yvals_1=noisyfunction(fun_1, 50, 0.1)

def fun_2(x):
    return x-0.5
xvals_2, yvals_2 = noisyfunction(fun_2, 50, 0.1)

## ...
```

In [ ]:
```

```