The vector version of the theorem

Theorem 4.1 (**Contraction Mapping Theorem**) *Suppose that D is a closed subset of $\mathbb{R}^n$, $\boldsymbol{g}\colon \mathbb{R}^n \to \mathbb{R}^n$ is defined on D, and $\boldsymbol{g}(D) \subset D$. Suppose further that $\boldsymbol{g}$ is a contraction on D in the ∞-norm. Then, $\boldsymbol{g}$ has a unique fixed point $\boldsymbol{\xi}$ in D, and the sequence $(\boldsymbol{x}^{(k)})$ defined by (4.3) converges to $\boldsymbol{\xi}$ for any starting value $\boldsymbol{x}^{(0)} \in D$.*

Theorem 1.3 (**Contraction Mapping Theorem**) *Let g be a real-valued function, defined and continuous on a bounded closed interval $[a, b]$ of the real line, and assume that $g(x) \in [a, b]$ for all $x \in [a, b]$. Suppose, further, that g is a contraction on $[a, b]$. Then, g has a unique fixed point ξ in the interval $[a, b]$. Moreover, the sequence (x_k) defined by (1.3) converges to ξ as $k \to \infty$ for any starting value x_0 in $[a, b]$.*

- Use of Gerschgorin ~~algorithm~~ Theorem to estimate eigenvalues.

Theorem 5.4 (**Gerschgorin's Theorem**) *Let $n \geq 2$ and $A \in \mathbb{C}^{n\times n}$. All eigenvalues of the matrix A lie in the region $D = \bigcup_{i=1}^{n} D_i$, where D_i, $i = 1, 2, \dots, n$, are the Gerschgorin discs of A defined by (5.17), (5.18).*

Theorem 5.5 (**Gerschgorin's Second Theorem**) *Let $n \geq 2$. Suppose that $1 \leq p \leq n-1$ and that the Gerschgorin discs of the matrix $A \in \mathbb{C}^{n\times n}$ can be divided into two disjoint subsets $D^{(p)}$ and $D^{(q)}$, containing p and $q = n - p$ discs respectively. Then, the union of the discs in $D^{(p)}$ contains p of the eigenvalues, and the union of the discs in $D^{(q)}$ contains $n - p$ eigenvalues. In particular, if one disc is disjoint from all the others, it contains exactly one eigenvalue, and if all the discs are disjoint then each disc contains exactly one eigenvalue.*

Computation example of least squares.

$$F_c(x) = c_1\varphi_1(x) + c_2\varphi_2(x) + c_3\varphi_3(x)$$

$\varphi_1(x) = 1$, $\varphi_2(x) = \cos x$, $\varphi_3(x) = \sin x$

First we need data to fit.

```
julia> phi1(x)=1.0
phi1 (generic function with 1 method)

julia> phi2(x)=cos(x)
phi2 (generic function with 1 method)

julia> phi3(x)=sin(x)
phi3 (generic function with 1 method)

julia> F(c,x)=[phi1(x),phi2(x),phi3(x)]'*c
F (generic function with 1 method)
```

Suppose some data

x	y
x_1	y_1
x_2	y_2
x_3	y_3
$\vdots$	$\vdots$
x_m	y_m

```
julia> xs=4*pi*rand(20)
20-element Vector{Float64}:
  9.547753029274727
  1.8217677360589428
  1.6031398306494404
  5.39404377674073
  6.480271494426944
  3.8525145941307812
  9.921444466433005
 10.442139736363245
  7.665549800539033
  6.767274193989993
  2.7794758027289386
  3.5155382491311022
  2.6123880141058966
  0.8751636047303569
  4.789863806635315
 11.718128975781715
  3.9460628804491304
  4.603146170508093
  1.067992793231892
  0.8460986170500502
```

← randomly create 20 values of x for testing...

```
julia> c=[2,3,4]
3-element Vector{Int64}:
 2
 3
 4
```

← actual values of c that the data comes from.

After generating the data we'll try to solve for c pretending that we didn't already know it.

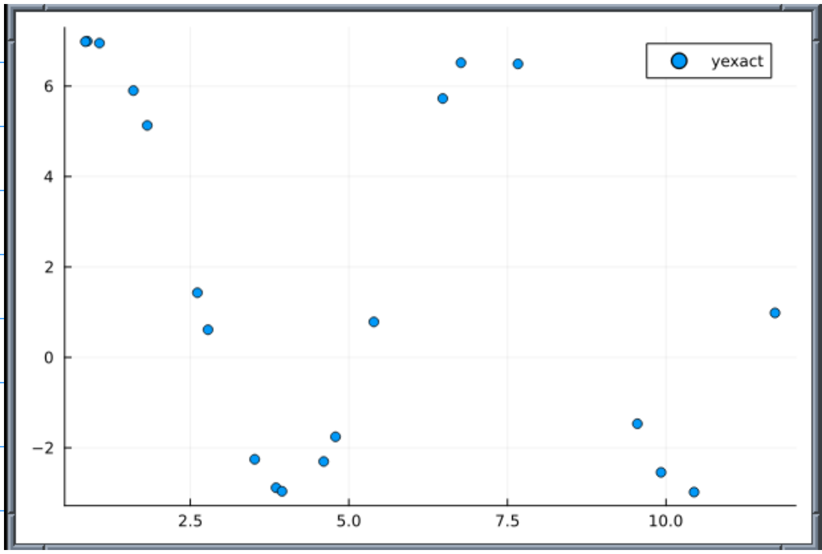
Turns the function of 2 variables into a function of one holding c constant

```
julia> yexact=(t->F(c,t)).(xs)
20-element Vector{Float64}:
 -1.4680056594149251
  5.129651377762678
  5.900894382756249
  0.7841121402641238
  5.725174949016598
 -2.883412439245213
 -2.5435173911384745
 -2.9797337435308844
  6.491152910570654
  6.5169065594741244
  0.6115713400946239
 -2.253844689030923
  1.4297611183229493
  6.9932072758385
 -1.7558092730294792
  0.9834344205494525
 -2.962332443946201
 -2.3032327113282696
  6.950592067375558
  6.9835268334475185
```

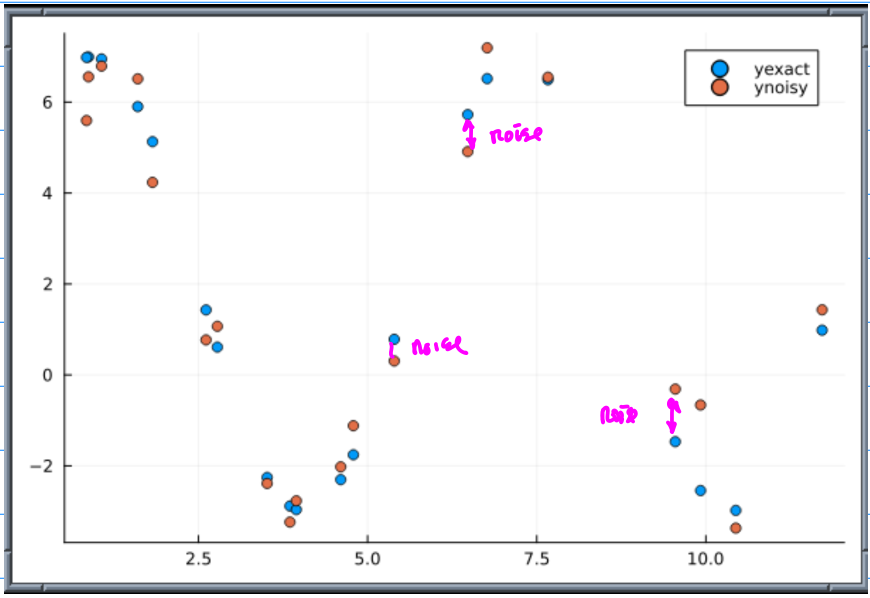
apply that function pointwise to each of the xs.

```
julia> using Plots
```

```
julia> scatter(xs,yexact,label="yexact")
```




```
julia> ynoisy=yexact+0.5*randn(20);

julia> scatter!(xs,ynoisy,label="ynoisy")
```



```
julia> phi=[phi1,phi2,phi3]
3-element Vector{Function}:
 phi1 (generic function with 1 method)
 phi2 (generic function with 1 method)
 phi3 (generic function with 1 method)

julia> V=[phi[j](xs[i]) for i=1:20,j=1:3]
20×3 Matrix{Float64}:
 1.0  -0.992448    -0.122665
 1.0  -0.248345     0.968672
 1.0  -0.0323379    0.999477
 1.0   0.630079    -0.776531
 1.0   0.980641     0.195813
 1.0  -0.757761    -0.652533
 1.0  -0.879176    -0.476497
 1.0  -0.525612    -0.850724
 1.0   0.187319     0.982299
 1.0   0.885099     0.465402
 1.0  -0.935149     0.354255
 1.0  -0.930893    -0.365291
 1.0  -0.863209     0.504847
 1.0   0.640871     0.767648
 1.0   0.0773973   -0.997
 1.0   0.661303    -0.750119
 1.0  -0.693493    -0.720463
 1.0  -0.109026    -0.994039
 1.0   0.481884     0.876235
 1.0   0.662909     0.7487
```

The Vandermonde Matrix.

$$V = \underbrace{\begin{bmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_n(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \cdots & \varphi_n(x_n) \\ \vdots & & & \\ \varphi_1(x_m) & \varphi_2(x_m) & & \varphi_n(x_m) \end{bmatrix}}_{[\varphi_j(x_i)]_{i,j}}$$

column

row.

Try to minimize

$$\| V c - y_{noisy} \|$$

```
julia> cnoisy=V\ynoisy
3-element Vector{Float64}:
 2.009960548979548
 2.6944678358542005
 3.773818116138601
```

The value of c for which the noisy date was most likely to have come from F_c.

behind the scenes uses QR factorization to solve the least squares problem $A = \tilde{Q}\tilde{R}$

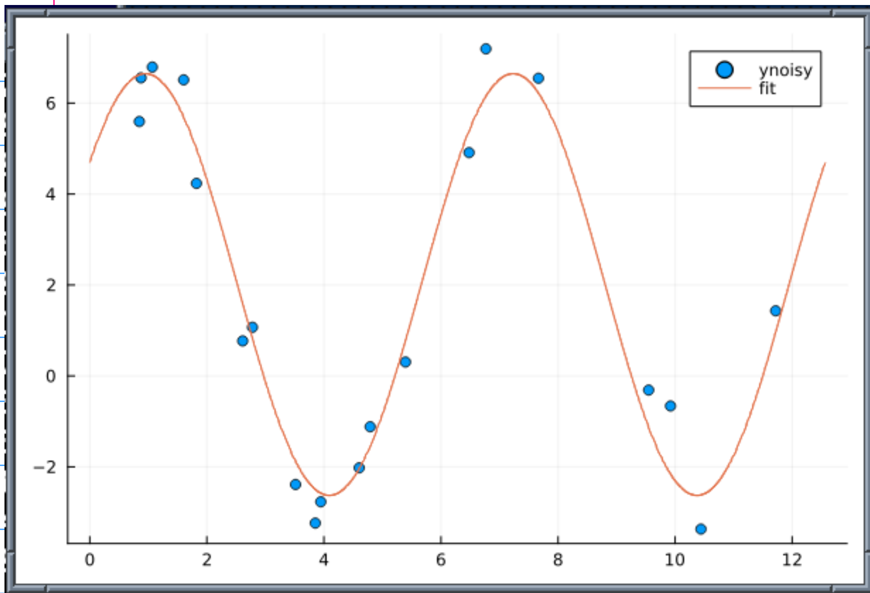
Householder reflectors to find Q

$$\tilde{R} c = \tilde{Q}^T y$$

solve by backsubstitution since $\tilde{R}$ is triangular

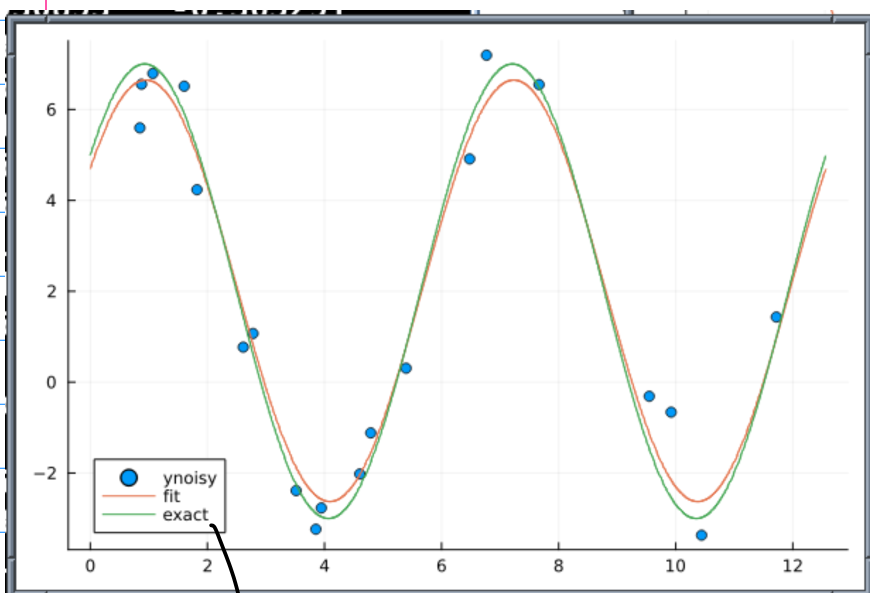
Check by drawing a graph

```
julia> xdom=0:0.01:4*pi
0.0:0.01:12.56

julia> yrange=(t->F(cnoisy,t)).(xdom);

julia> plot!(xdom,yrange)
```



approximate linear model

How did I do in comparison to the actual model



$F_{(2,3,4)}(x)$

Note the data was generated by the green model but is actually closer to the red one.

What happens without the noise?

noisy data

```
julia> cnoisy=V\ynoisy
3-element Vector{Float64}:
 2.009960548979548
 2.6944678358542005
 3.773818116138601
```

$Vc = y_{exact}$ over determined system.. $y_{exact} \in C(V)$.

```
julia> cexact=V\yexact
3-element Vector{Float64}:
 1.9999999999999996
 3.0000000000000004
 3.999999999999999
```

$\left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} \begin{matrix} 2 \\ 3 \\ 4 \end{matrix}$

Up to rounding.

How good is the fit. I.e. what is the minimum

$$\| Vc - y_{noisy} \| \approx 3.07\ldots$$

Check

```
julia> using LinearAlgebra

julia> norm(V*cnoisy-ynoisy)
3.0717214650120424
```

Small it is the better the fit... but we don't expect it to fit exactly because of the noise

---

HW5 will be about least squares... please check in a day for the assignment...