04_best_approximation

October 16, 2019

1 Best Approximation in Hilbert Spaces

[1]:
```
%matplotlib inline
import sympy as sym
import pylab as pl
import numpy as np
import numpy.polynomial.polynomial as n_poly
import numpy.polynomial.legendre as leg
```

1.1 Mindflow

We want the best approximation (in Hilbert Spaces) of the function f, on the space $V = \text{span}\{v_i\}$. Remember that $p \in V$ is best approximation of f if and only if:

$$(p - f, q) = 0, \quad \forall q \in V.$$

Focus one second on the fact that both p and q belong to V. We know that any q can be expressed as a linear combination of the basis functions v_i:

$$(p - f, v_i) = 0, \quad \forall v_i \in V.$$

Moreover p is uniquely defined by the coefficients p^j such that $p = p^j\, v_j$. Collecting this information together we get:

$$(v_j, v_i)p^j = (f, v_i), \quad \forall v_i \in V.$$

Now that we know our goal (finding these p^j coefficients) we do what the rangers do: we explore!

We understand that we will need to invert the matrix:

$$M_{ij} = (v_j, v_i) = \int v_i \cdot v_j$$

What happens if we choose basis functions such that $(v_j, v_i) = \delta_{ij}$?

How to construct numerical techniques to evaluate integrals in an efficient way?

Evaluate the L^2 projection.

1.2 Orthogonal Polynomials

Grham Schmidt

$$p_0(x) = 1, \qquad p_k(x) = x^k - \sum_{j=0}^{k-1} \frac{(x^k, p_j(x))}{(p_j(x), p_j(x))}$$

or, alternatively

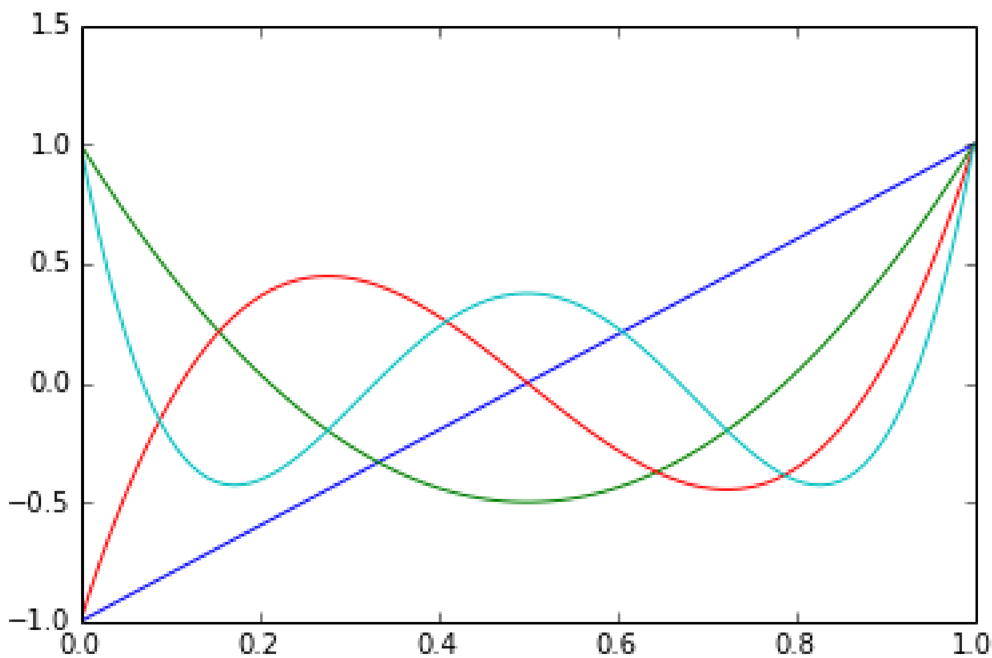
$$p_0(x) = 1, \qquad p_k(x) = x\, p_{k-1}(x) - \sum_{j=0}^{k-1} \frac{(x p_{k-1}(x), p_j(x))}{(p_j(x), p_j(x))}$$

[2]:
```
def scalar_prod(p0,p1,a=0,b=1):
    assert len(p0.free_symbols) <= 1, "I can only do this for single variable␣
 ↪functions..."
    t = p0.free_symbols.pop() if len(p0.free_symbols) == 1 else sym.symbols('t')
    return sym.integrate(p0*p1,(t,a,b))
```

[4]:
```
t = sym.symbols('t')
k = 3

Pk = [1+0*t] # Force it to be a sympy expression

for k in range(1,5):
    s = 0
    for j in range(0,k):
        s+= scalar_prod(t**k,Pk[j])/scalar_prod(Pk[j],Pk[j])*Pk[j]
    pk = t**k-s
    # pk = pk/sym.sqrt(scalar_prod(pk,pk))
    pk = pk/pk.subs(t,1.)
    Pk.append(pk)


M = []
for i in range(len(Pk)):
    row = []
    for j in range(len(Pk)):
        row.append(scalar_prod(Pk[i],Pk[j]))
    M.append(row)

M = sym.Matrix(M)

print(M)
```

```
x = np.linspace(0,1,100)

for p in Pk:
    if p != 1 :
        fs = sym.lambdify(t, p, 'numpy')
        #print x.shape
        #print fs(x)
        _ = pl.plot(x,fs(x))
```

```
Matrix([[1, 0, 0, 1.77635683940025e-15, -5.32907051820075e-15], [0,
0.333333333333333, -8.88178419700125e-16, 0, -7.10542735760100e-15], [0,
-8.88178419700125e-16, 0.200000000000003, 2.84217094304040e-14,
-5.68434188608080e-14], [1.77635683940025e-15, 0, 2.84217094304040e-14,
0.142857142857110, -3.26849658449646e-13], [-5.32907051820075e-15,
-7.10542735760100e-15, -5.68434188608080e-14, -3.26849658449646e-13,
0.111111111112791]])
```



1.3 Theorem

Le q be nonzero polynomial of degree $n+1$ and $\omega(x)$ a positive weight function, s. t.:

$$\int_a^b x^k q(x)\,\omega(x) = 0, \quad k = 0, \dots, n$$

If x_i are zeros of $q(x)$, then:

$$\int_a^b f(x)\,\omega(x) \approx \sum_{i=0}^n w_i\, f(x_i)$$

with:

$$w_i = \int_a^b l_i(x)\,\omega(x)$$

is exact for all polynomials of degree at most $2n+1$. Here $l_i(x)$ are the usual Lagrange interpolation polynomials.

Proof: assume $f(x)$ is a polynomial of degree at most $2n+1$ and show:

$$\int_a^b f(x)\,\omega(x) = \sum_{i=0}^n w_i\, f(x_i).$$

Usign the polynomial division we have:

$$\underbrace{f(x)}_{2n+1} = \underbrace{q(x)}_{n+1}\,\underbrace{p(x)}_{n} + \underbrace{r(x)}_{n}.$$

By taking x_i as zeros of $q(x)$ we have:

$$f(x_i) = r(x_i)$$

Now:

$$\begin{aligned}\int_a^b f(x)\,\omega(x) &= \int_a^b [q(x)\,p(x) + r(x)]\,\omega(x) \\ &= \underbrace{\int_a^b q(x)\,p(x)\,\omega(x)}_{=0} + \int_a^b r(x)\,\omega(x)\end{aligned}$$

Since $r(x)$ is a polynomial of order n this is exact:

$$\int_a^b f(x)\,\omega(x) = \int_a^b r(x)\,\omega(x) = \sum_{i=0}^n w_i\, r(x_i)$$

But since we chose x_i such that $f(x_i) = r(x_i)$, we have:

$$\int_a^b f(x)\,\omega(x) = \int_a^b r(x)\,\omega(x) = \sum_{i=0}^n w_i\, f(x_i)$$

This completes the proof.

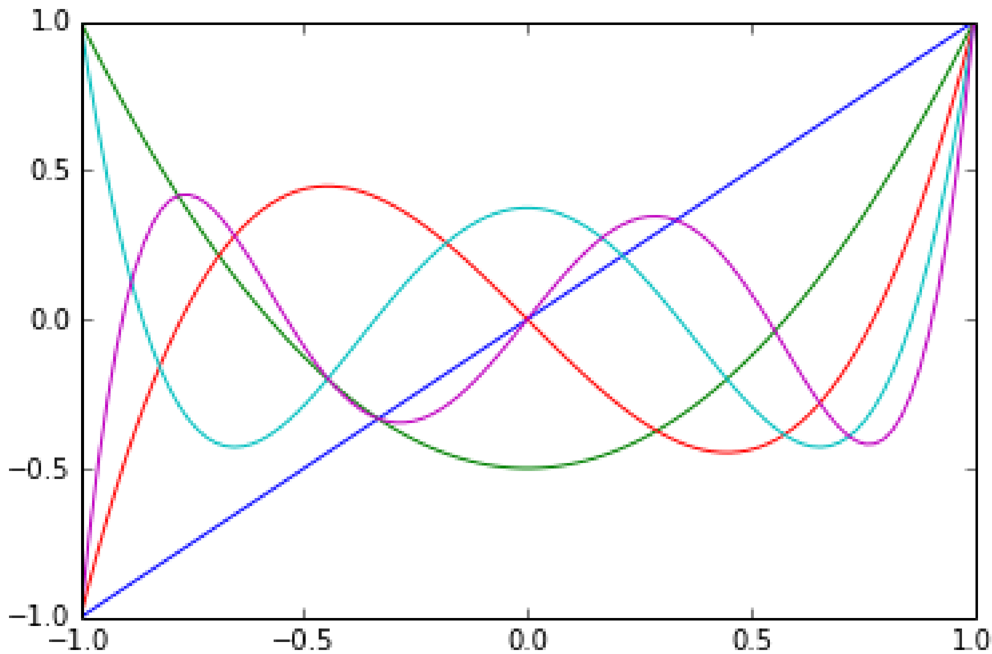
1.4 Legendre Polynomial

Two term recursion, to obtain the same orthogonal polynomials above (defined between [-1,1]), normalized to be one in $x = 1$:

$$(n+1)p^{n+1}(x) = (2n+1)\,x\,p^n(x) - n\,p^{n-1}(x)$$

[5]:
```
Pn = [1.,t]

#Pn = [1.,x, ((2*n+1)*x*Pn[n] - n*Pn[n-1])/(n+1.) for n in range(1,2)]

for n in range(1,5):
    pn1 = ((2*n+1)*t*Pn[n] - n*Pn[n-1])/(n+1.)
    Pn.append(sym.simplify(pn1))

print(Pn)

#print(sym.poly(p))
#print(sym.real_roots(sym.poly(p)))
print(sym.integrate(Pn[4]*Pn[3],(t,-1,1)))


x = np.linspace(-1,1,100)
for p in Pn:
    if p != 1. :
        fs = sym.lambdify(t, p, 'numpy')
        #print x.shape
        #print fs(x)
        _ = pl.plot(x,fs(x))
```

```
[1.0, t, 1.5*t**2 - 0.5, t*(2.5*t**2 - 1.5), 4.375*t**4 - 3.75*t**2 + 0.375,
t*(7.875*t**4 - 8.75*t**2 + 1.875)]
0
```



In our proof we selected to evaluate x_i at the zeros of the legendre polynomials, this is why we need to evaluate the zeros of the polynomials.

[6]:
```
print(sym.real_roots(sym.poly(Pn[2])))

#q = [-1.]+sym.real_roots(sym.poly(Pn[2]))+[1.]
q = sym.real_roots(sym.poly(Pn[3]))
print(q)

#for p in Pn:
#    if p != 1. :
#        #print(sym.poly(p))
#        #print(sym.real_roots(sym.poly(p)))
#        print(sym.nroots(sym.poly(p)))
```

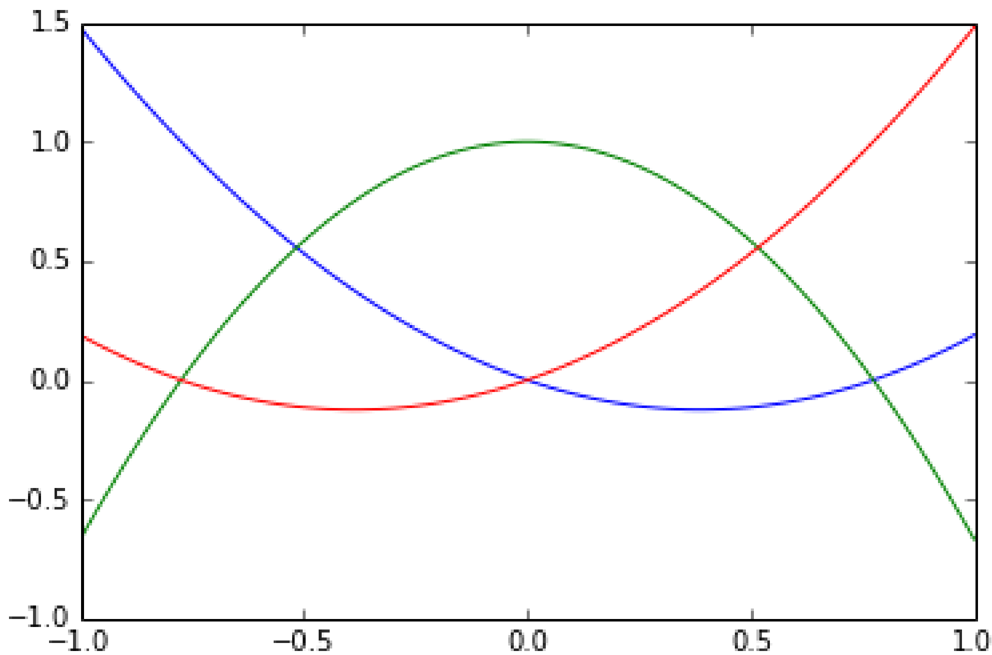
```
[-sqrt(3)/3, sqrt(3)/3]
[-sqrt(15)/5, 0, sqrt(15)/5]
```

$$w_i = \int_{-1}^{1} l_i(x)$$

[7]:
```
Lg = [1. for i in range(len(q))]
print(Lg)

#for i in range(n+1):
```

```
for i in range(len(q)):
    for j in range(len(q)):
        if j != i:
            Lg[i] *= (t-q[j])/(q[i]-q[j])

print(Lg)

x = np.linspace(-1,1,100)

for l in Lg:
    fs = sym.lambdify(t, l, 'numpy')
    _ = pl.plot(x,fs(x))
```

```
[1.0, 1.0, 1.0]
[0.833333333333333*t*(t - sqrt(15)/5), -1.66666666666667*(t - sqrt(15)/5)*(t +
sqrt(15)/5), 0.833333333333333*t*(t + sqrt(15)/5)]
```



[8]:
```
for l in Lg:
    print(sym.integrate(l,(t,-1,1)))
```

```
0.555555555555555
0.888888888888889
0.555555555555555
```

1.4.1 Hint

Proiezione usando polinomi LEGENDRE (f,v_i)

2 Now let's get Numerical

From now on I work on the $[0, 1]$ interval, becouse i like it this way :)

In the previus section we explored what sympbolically was happening, now we implement things on the computer. We saw how important are the legendre plynomials. Here a little documentation on that. I pont it out not because you need to read it all, but because I would like you get some aquitance with this criptic documentation pages doc.

The problem we aim at solving is finding the coefficents p_j such that:

$$(v_j, v_i)p^j = (f, v_i), \quad \forall v_i \in V.$$

Remind in this section the einstein notation holds.

We can expand the compact scalar product notation:

$$p^j \int_0^1 v_i\, v_j = \int_0^1 f\, v_i, \quad \forall v_i \in V.$$

We consider $V = \text{span}\{l_i\}$. Our problem becomes:

$$p^j \int_0^1 l_i\, l_j = \int_0^1 f\, l_i, \quad \text{for } i = 0, \ldots, \texttt{deg}$$

Let's focus on mass matrix:

$$\int_0^1 l_i(x)\, l_j(x) = \sum_k l_i(x_k)\, w_k\, l_j(x_k) =$$

$$= \begin{pmatrix} l_0(x_0) & l_0(x_1) & \ldots & l_0(x_q) \\ l_1(x_0) & l_1(x_1) & \ldots & l_1(x_q) \\ & \ldots & \ldots & \\ l_n(x_0) & l_n(x_1) & \ldots & l_n(x_q) \end{pmatrix} \begin{pmatrix} w_0 & 0 & \ldots & 0 \\ 0 & w_1 & \ldots & 0 \\ & \ldots & \ldots & \\ 0 & 0 & \ldots & w_q \end{pmatrix} \begin{pmatrix} l_0(x_0) & l_1(x_0) & \ldots & l_n(x_0) \\ l_0(x_1) & l_1(x_1) & \ldots & l_n(x_1) \\ & \ldots & \ldots & \\ l_0(x_q) & l_1(x_q) & \ldots & l_n(x_q) \end{pmatrix} = B\, W\, B^T$$

A piece of curiosity, how the the two functions to find theros in two different ways

[10]:
```
print sym.nroots(sym.poly(Pn[-1]))
coeffs = np.zeros(6)
coeffs[-1] = 1.
print(leg.legroots(coeffs))
```

```
[-0.906179845938664, -0.538469310105683, 0, 0.538469310105683,
0.906179845938664]
[ -9.06179846e-01  -5.38469310e-01  -5.96500148e-17   5.38469310e-01
   9.06179846e-01]
```

[11]:
```
print gauss_points(3)
print(np.sqrt(3./5.)*.5)+.5
```

```
[ 0.11270167  0.5         0.88729833]
0.887298334621
```

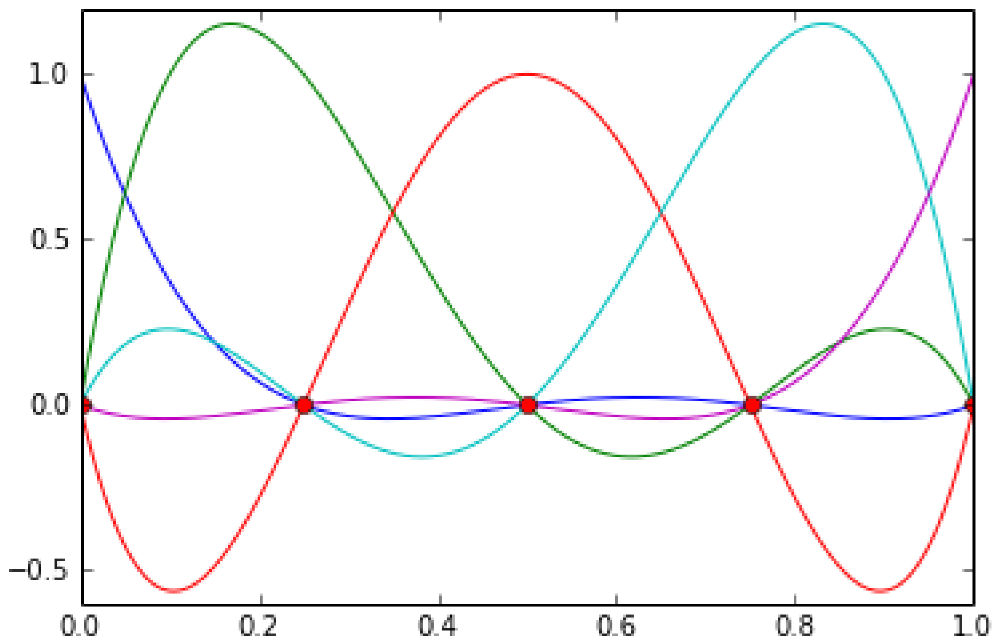
[12]:
```
def define_lagrange_basis_set(q):
    n = q.shape[0]
    L = [n_poly.Polynomial.fromroots([xj for xj in q if xj != q[i]]) for i in␣
 ↪range(n)]
    L = [L[i]/L[i](q[i]) for i in range(n)]
    return L
```

differenza fra le roots ”simboliche” e non

[14]:
```
deg = 4
Nq = deg+1
p,w = leg.leggauss(Nq)
w = .5 * w
p = .5*(p+1)
#print p
#print w
W = np.diag(w)
#print W
```

[15]:
```
int_p = np.linspace(0,1,deg+1)
L = define_lagrange_basis_set(int_p)

print(len(L))

x = np.linspace(0,1,1025)
for f in L:
    _ = pl.plot(x, f(x))
_ = pl.plot(int_p, 0*int_p, 'ro')
```

```
5
```



[16]:
```
B = np.zeros((0,Nq))
for l in L:
    B = np.vstack([B,l(p)])
```

Recall:

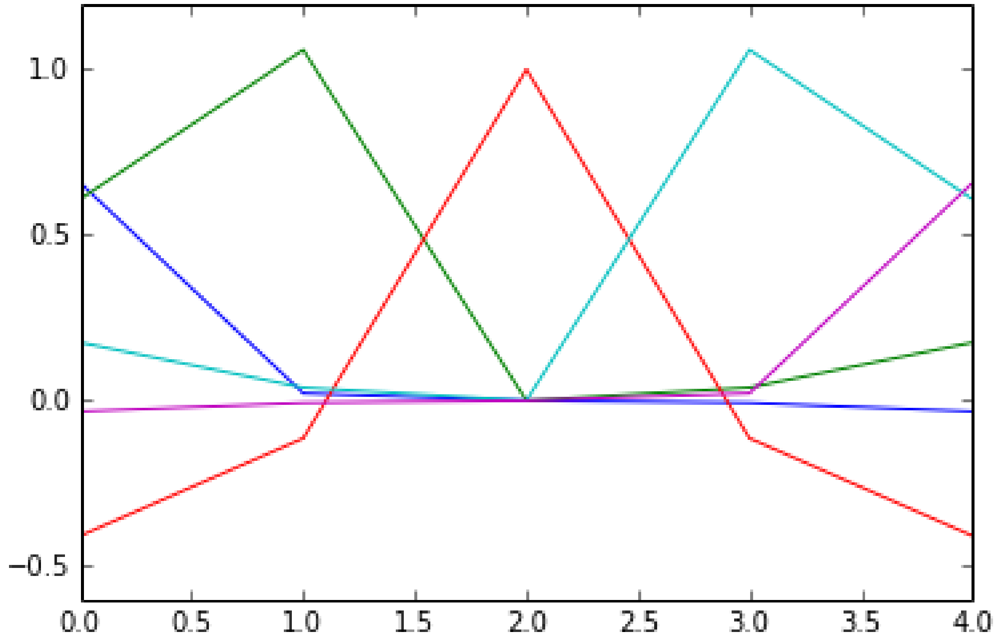
$$B\,W\,B^T p = BWf$$

$$B\,W\,B^T = \begin{pmatrix} l_0(x_0) & l_0(x_1) & \dots & l_0(x_q) \\ l_1(x_0) & l_1(x_1) & \dots & l_1(x_q) \\ & & \ddots & \\ l_n(x_0) & l_n(x_1) & \dots & l_n(x_q) \end{pmatrix} \begin{pmatrix} w_0 & 0 & \dots & 0 \\ 0 & w_1 & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & w_q \end{pmatrix} \begin{pmatrix} l_0(x_0) & l_1(x_0) & \dots & l_n(x_0) \\ l_0(x_1) & l_1(x_1) & \dots & l_n(x_1) \\ & & \ddots & \\ l_0(x_q) & l_1(x_q) & \dots & l_n(x_q) \end{pmatrix}$$

[17]:
```
print(B.shape)
_ = pl.plot(B.T)
M = B.dot(W.dot(B.T))
print np.linalg.matrix_rank(M)
print np.linalg.cond(M)
```

```
(5, 5)
[[  6.57727883e-01   2.20631033e-02  -8.88178420e-16  -6.61878610e-03
   -3.23726701e-02]
 [  6.07692695e-01   1.05879718e+00   1.77635684e-15   3.92223408e-02
    1.75534108e-01]
 [ -4.08582015e-01  -1.13463840e-01   1.00000000e+00  -1.13463840e-01
```

```
   -4.08582015e-01]
 [  1.75534108e-01   3.92223408e-02   4.44089210e-16   1.05879718e+00
    6.07692695e-01]
 [ -3.23726701e-02  -6.61878610e-03  -2.22044605e-16   2.20631033e-02
    6.57727883e-01]]
5
14.1390607069
```



[18]:
```
def step_function():
    def sf(x):
        index = where((x>.3) & (x<.7))
        step = zeros(x.shape)
        step[index] = 1
        return step
    return lambda x : sf(x)
```

$$B\,W\,f = \begin{pmatrix} l_0(x_0) & l_0(x_1) & \dots & l_0(x_q) \\ l_1(x_0) & l_1(x_1) & \dots & l_1(x_q) \\ & \dots & \dots & \\ l_n(x_0) & l_n(x_1) & \dots & l_n(x_q) \end{pmatrix} \begin{pmatrix} w_0 & 0 & \dots & 0 \\ 0 & w_1 & \dots & 0 \\ & \dots & \dots & \\ 0 & 0 & \dots & w_q \end{pmatrix} \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_q) \end{pmatrix}$$

[19]:
```
g = lambda x: np.sin(2*np.pi*x)
#g = step_function()
p = p.reshape((p.shape[0],1))
```

```
G = g(p)
print G.shape
print B.shape
print W.shape
G = B.dot(W.dot(G))
```

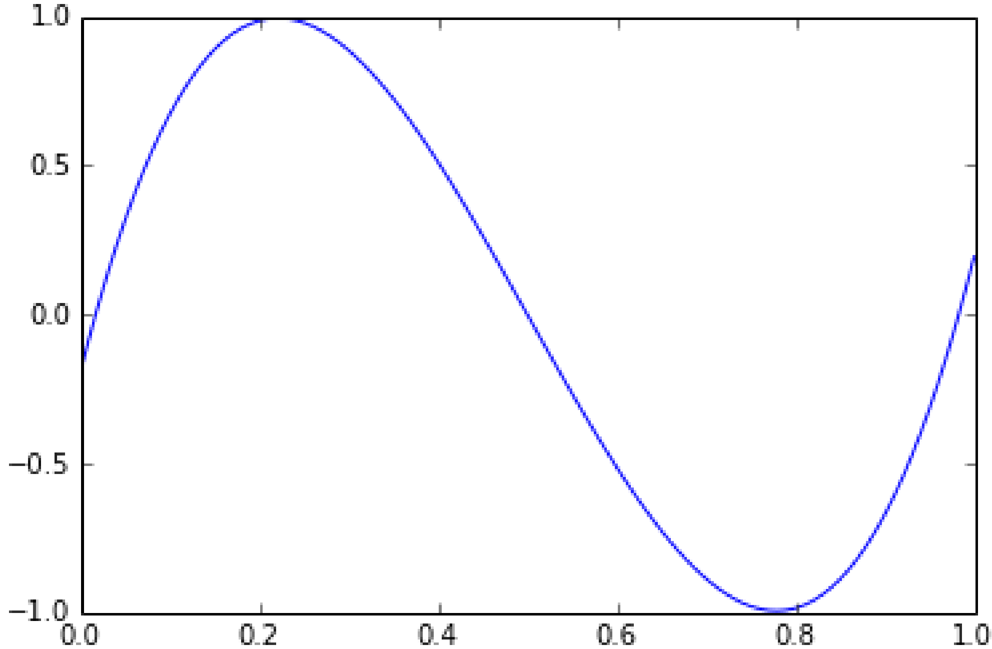
```
(5, 1)
(5, 5)
(5, 5)
```

[20]:
```
u = np.linalg.solve(M, G)
print u
```

```
[[ -1.92161045e-01]
 [  9.79052672e-01]
 [ -1.35712301e-15]
 [ -9.79052672e-01]
 [  1.92161045e-01]]
```

[21]:
```
def get_interpolating_function(LL,ui):
    def func(LL,ui,x):
        acc = 0
        for L,u in zip(LL,ui):
            #print(L,u)
            acc+=u*L(x)
        return acc
    return lambda x : func(LL,ui,x)
```

[22]:
```
I = get_interpolating_function(L,u)
sampling = np.linspace(0,1,101)
_= pl.plot(sampling, I(sampling))
#plot(xp, G,'ro')
```



2.1 Diference in between projection and interpolation runge example

Proiezione usando polinomi LEGENDRE (f,v_i) con quadratura con 18 punti

Interpolazione usando polinomi LAGRANGE (sui punti di quadratura che sono i punti di gauss della funzione sopra)

[ ]:

```

```