Orthogonal Polynomials

and
Gauss Quadrature Formulae

The subject of orthogonal polynomial is rather self contained and can be studied in the classical treatise by G. Szegö, Orthogonal Polynomials, American Mathematical Society, Providence R.I. 1959. For our goals much less is needed, you may find an account (in Italian) on the page www.eoinfnpr.it looking *for* MMF.pdf, pag.69. Here we recall basic facts, show how OP are useful in building Gauss quadrature schemes and provide `matlab` codes where these ideas are implemented.

A sequence of polynomials $\{p_0(x), p_1(x), \ldots, p_n(x)\ldots\}$ in the real variable x is called a family of orthogonal polynomials with respect to the *measure* $w(x)$ if 1) $p_n(x)$ is of n-th degree for every n and

2) $$\int_a^b w(x)\, p_i(x)\, p_j(x) dx = \delta_{ij}$$

It is possible to write explicitly an expression for $p_n(x)$ in terms of the *moments* $m_i \equiv \int_a^b w(x)\, x^i\, dx$, but this is not relevant here (see the references already cited). The crucial property for what concerns the quadrature formula discovered by C.F. Gauss is the following: there exists a recurrence relation of the form

$$x\, p_j(x) = a_j\, p_{j+1}(x) + b_j\, p_j(x) + c_j\, p_{j-1}(x)$$

The coefficients a, b, c which define the recurrence and that permit to build the whole sequence starting from $p_0(x) = \mathcal{N}_0$ are easily computed from the normalization of the polynomials. No details here. Just consider the fact that the recurrence relation can

be expressed as *(Golub and Welsch, G.H. Golub and J.H. Welsch, Calculation of gauss quadrature rules, Math. Comput. 23 (1969))*

$$\begin{pmatrix} b_0 & a_0 & \cdots & & & & \\ c_1 & b_1 & a_1 & \cdots & & & \\ . & c_2 & b_2 & a_2 & \cdots & & \\ \cdots & \ddots & \ddots & \ddots & \cdots & & \\ \cdots & & c_k & b_k & a_k & \cdots & \\ & & & \ddots & \ddots & \ddots & \cdots \\ \cdots & & & & c_{n\text{-}1} & b_{n\text{-}1} & a_{n\text{-}1} \\ \cdots & & & & & c_n & b_n \end{pmatrix} \begin{pmatrix} P_0(x) \\ P_1(x) \\ P_2(x) \\ \vdots \\ P_k(x) \\ \vdots \\ P_{n\text{-}1}(x) \\ P_n(x) \end{pmatrix} = x \begin{pmatrix} P_0(x) \\ P_1(x) \\ P_2(x) \\ \vdots \\ P_k(x) \\ \vdots \\ P_{n\text{-}1}(x) \\ P_n(x) \end{pmatrix} - a_n \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \\ P_{n+1}(x) \end{pmatrix}$$

In this form the recurrence relation is telling us that the tridiagonal matrix on the left built with the recurrence coefficients has the zeroes of the polynomial $P_{N+1}(x)$ as eigenvalues and correspondingly we get the eigenvectors in terms of the evaluation of lower degree polynomials {0,1,…,n}. Diagonalizing a tridiagonal matrix is numerically very cheap and fast. So we can take for granted that we know all eigenvalues and eigenvectors. What for? Consider the integral

$$\int w(x)\, F(x)\, dx$$

for a polynomial F of degree $2n$-1. Let's define $\phi(x) = \sum_{j=1}^{n} F(x_j) \frac{P_n(x)}{(x - x_j) P_n'(x)}$. At the zeroes of $P_n(x)$ we have $\phi(x_j) \equiv F(x_j)$ then. it must hold $\phi(x) \equiv F(x) + P_n(x)\, Q(x)$ for some polynomial Q of degree n-1. By integrating F we then obtain

$$\int_a^b F(x)\,w(x)\,dx = \int_a^b w(x)\sum_j F(x_j)\frac{P_n(x)}{(x-x_j)P_n'(x)}dx + \int_a^b P_n(x)\,Q(x)\,w(x)\,dx$$

BUT Q is orthogonal to $P_n(x)$, hence we get

$$\int_a^b F(x)\,w(x)\,dx = \sum_j F(x_j)\,\frac{1}{P_n'(x)}\int_a^b \frac{w(x)}{x-x_j}P_n(x)\,dx$$

We see that the integral of F(x) is represented bby a weighted sum of its values at the zeros of $P_n(x)$. The weights (*Gaussian weights*) can be computed once for all and used in any situation. If *F(x)* is a polynomial the formula is exact, if *F(x)* any analytic function around the real axis the formula will be only approximate, but the approximation is very good even for moderate values of *n* . The good news are that the weights

$$\lambda_j \equiv \int_a^b \frac{w(x)}{x-x_j}\,\frac{P_n(x)}{P_n'(x)}\,dx$$

can be computed in terms of the eigenvectors of the original tridiagonal matrix! Details to be found on the previous references.

These are the general ideas. The practical implementation in matlab is quite easy. It can be found in the included package GAUSQUAD for the whole set of *classical polynomials,* Hermite, Jacobi, Legendre, Laguerre, Tchebychev I and II kind.
For a professional approach to the problem of Gauss quadrature with an eye to recent developments you are advise to consider the beautiful package CHEBFUN *which can be freely downloaded from* www.chebfun.org.

The main matab program is Gaussquad.m:

```
function q = Gaussquad(f,n,meas,opt)
% Driver program for Gaussian integration over
% classical polynomials
% Usage:
%         gausquad('func',Npts,'type',[options])
% where
% 'func' is the name of the function to integrate
% Npts = number of Gaussian points
% 'meas'= one of 'her' (Hermite)
%                'leg' (Legendre)
%                'lag' (Laguerre) (opt=a for the measure x^a e^(-x))
%                'jac' (Jacobi)   (opt=[a,b]  "    "  (1-x)^a (1+x)^b
%                'geg' (Gegenbauer) (opt=l for the measure (1-x^2)^(l-1/2)
%
% Try help gaussleg, gausslag, gaussherm, gaussgeg, gaussjac
%%%
```

The idea is: you have to compute the integral of a given function *f(x)* against a measure which is in the list of the classical polynomials, e.g.

$$\int_{-\infty}^{\infty} \sin(x^2)\, \exp\{-x^2\}\, dx$$

then you can proceed as follows:

```
>> f = @(x) sin(x.^2);
>> q=Gaussquad(f,1024,'her');
>> ans =
     0.321797126452791
```

which is correct to 15 figures *(*exact result is *sin(pi/8)/2^(1/4))*. The function to be integrate can be represented as an anonymous function in matlab, if it is simple enough, or by defining an external function in a file fun.m and calling it with its pointer, i.e.

```
>>  Gaussquad(@fun,256,'her');
```

You have all these standard measure available:

Hermite: $\exp(-x^2)/\sqrt{\pi}$ interval $(-\infty,\infty)$
Legendre: 1/2 " (-1,1)
Laguerre: $x^\alpha \exp\{-x\}$ " (0,∞)
Tchebychev I: $(1-x^2)^{-1/2}$ " (-1,1)
Tchebychev II: $\sqrt{1-x^2}$ " (-1,1)
Jacobi: $(1-x)^a(1+x)^b$ " (-1,1)

The various codes do the following: build the tridiagonal matrix out of the recurrence coefficients, diagonalize it with *eig*, and compute the Gaussian

weights out of the matrix containing the information on the eigenvectors. Typically:

```
function [x,w,p] = gausherm(n)
% HERMITE POLYNOMIALS: zeroes and gaussian integration coefficients.
% Usage:
%    [x,w,p] = gausherm(n)
%    returns p = H_j(x_k), j,k = 0:n-1 (orthonormal)
%          x roots of H_{n}
%          w Gauss integration weights for the measure
%          \int exp(-x^2) dx/sqrt(pi)
%
%   Ex. sum(w.*x.^10) =  10!/2^5/5! * (1/2)^5 = 29.53125999999 = 945/32
%   Ex. sum(w.*exp(2*i*x) = .36787944117144 = exp(-1)
%
disp('Building the companion matrix of H_n(x)');
disp(' ');

l1 = sqrt(1:n-1)/sqrt(2);
D = diag(l1,-1)+diag(l1,1);

disp('             ... and diagonalizing it:');
disp(' ');

[U,X]  = eig(D);
[x,ind]= sort(-diag(X)');

U = U(:,ind);
```

```
disp('Build Hermite polynomials and Gauss weights');
disp(' ');

p0 = U;
w = p0(1,:).^2;
p = U/diag(p0(1,:));
end
```

Subsequently

```
    F=feval(f,x);
    q=sum(w.*F);
```

computes the approximate integral.