**Project 1: Part 2**

Here we continue the preliminary project.

### Calculus on Polynomials

There's a certain pattern in defining many generic functions:

```
(define-generic (do-something (f))
  ;; define it for appropriate non-Meroon objects, numbers perhaps
  ;; throw an error if do-something is not appropriate for f
  )
(define-method (do-something (f Polynomial))
  ;; define do-something for Polynomials.
  ;; apply do-something-to-termlist to (Polynomial-terms f)
  ;; to implement do-something on Polynomials
  )
(define (do-something-to-termlist terms)
  ;; go through the terms of the polynomial,
  ;; calling do-something-to-term on each term and
  ;; accumulating the result as appropriate
  )
(define (do-something-to-term term)
  ;; basic operation of do-something on one polynomial term
  )
```

Following this patterm, define generic functions

```
(define-generic (differentiate (f) variable)
  (if (number? f)
      0
      (error "differentiate: argument not of correct type " f)))
(define-generic (integrate (f) variable #!optional (a #f) (b #f))
  (if (number? f)
      (if (and (number? a)
       (number? b))
  (multiply f (subtract b a))
  (instantiate Polynomial
       variable: variable
       terms: (adjoin-term (make-term 1 f)
   (the-empty-termlist))))
      (error "integrate: unknown argument type " f variable)))
```

and then define appropriate methods for Polynomials. In integrate `a` and `b` are the optional two endpoints; if they aren't given return an indefinite integral, if they are given, return a definite integral, as such:

```
(define-method (integrate (p Polynomial) variable #!optional (a #f) (b #f))
```

```
    (if (Polynomial-variable= (Polynomial-variable p)
      variable)
        (let ((indefinite-integral
       (instantiate Polynomial
    variable: variable
    terms: (integrate-termlist (Polynomial-terms p)))))
  (if (and (number? a)
   (number? b))
      (subtract (evaluate indefinite-integral b)
        (evaluate indefinite-integral a))
      indefinite-integral))
        (error "integrate: The variable of integration is not the variable of the polyno-
mial " p variable)))
```

(At this point I'm wondering whether just carrying around all these `variable`s; they just seem to get in the way, and if we think of polynomials as symbolic expressions, they're OK, but if we think of polynomials as functions of a certain type, they just get in the way. SICP is treating them as symbolic expression.)

### Orthogonal polynomials

Now we can define inner products:

```
(define (make-inner-product weight variable left right)
  (lambda (p q)
    (integrate (multiply p (multiply q weight))   ;; weight can be a constant
       variable left right)))
```

This function takes four arguments and itself returns a function of two arguments:

$$\int_a^b p(\texttt{variable})\, q(\texttt{variable})\, w(\texttt{variable})\, d\texttt{variable} = \langle p, q\rangle.$$

Given an inner product, the recursion for orthogonal polynomials is

$$\begin{gathered}
P_{-1}(x) = 0; \quad P_0(x) = 1; \\
S_i = \langle P_i(x), P_i(x)\rangle,\ B_i = \frac{\langle xP_i(x), P_i(x)\rangle}{S_i} \\
C_i = \begin{cases} \text{arbitrary}, & i = 0, \\ \dfrac{S_i}{S_{i-1}}, & i > 0 \end{cases} \\
P_{i+1}(x) = (x - B_i)P_i(x) - C_i P_{i-1}(x), \quad i = 0, 1, 2, \dots .
\end{gathered}$$

See Conte and de Boor, *Elementary Numerical Analysis,* third edition, page 254. (We take $A_i = 1$ for all $i$.)

We define the Gauss-Lobatto weight and inner product on (-1,1):

```
;;; The Gauss-Lobatto weight on (-1, 1)
(define (G-L-weight variable)
  ;; 1-x^2=-(x^2-1) (we can only subtract constants on right)
  (let ((X (variable->Polynomial variable)))
```

```
    (negate (subtract (multiply X X) 1))))
(define (G-L-inner-product variable left right)
  (make-inner-product (G-L-weight variable) variable left right))
```

See Hämmerlin and Hoffmann, *Numerical Mathematics*, page 302.

Write a function

```
(define (make-orthogonal-polynomials inner-product variable n)
  ;; fill in the blanks
  )
```

that calculates $P_0$, $P_1$, … , $P_n$ given an inner product and a variable. You should be able to do something like this:

```
euler-6% gsi++
[ Meroon V3 Paques2001+1 $Revision: 1.2 $ ]
Gambit v4.1.2
> (load "all")
"/export/users/lucier/programs/615project/2007/project-1/all.scm"
> (define weight (G-L-weight 'x))
> (define inner-product (G-L-inner-product 'x))
> (define ps (make-orthogonal-polynomials inner-product 'x 10))
> (for-each show ps)
x^10-15/7x^8+30/19x^6-150/323x^4+15/323x^2-3/4199
x^9-36/19x^7+378/323x^5-84/323x^3+63/4199x
x^8-28/17x^6+14/17x^4-28/221x^2+7/2431
x^7-7/5x^5+7/13x^3-7/143x
x^6-15/13x^4+45/143x^2-5/429
x^5-10/11x^3+5/33x
x^4-2/3x^2+1/21
x^3-3/7x
x^2-1/5
x
1
0
```

Now we need to find the zeros $x_{n\kappa}$ of $P_n(x)$. One of the best (the stablest, the most accurate) ways to find the zeros of a polynomial

$$P(x) = x^n + p_{n-1}x^{n-1} + p_{n-2}x^{n-2} + \cdots + p_1 x + p_0$$

is to use `dgeev.f` from LAPACK to compute the eigenvalues of the matrix

$$\begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ & & \vdots & \ddots & \\ -p_0 & -p_1 & -p_2 & \dots & -p_{n-1} \end{pmatrix}.$$

There's no point to rewriting `dgeev.f` in Scheme, so we should use a so-called *Foreign Function Interface* (FFI) to call Fortran functions from Scheme. FFIs aren't standardized, but Gambit has one. (You run into the same problem calling functions defined in one language from functions in another language.)

I thought I could compile dgeev.f and its dependencies and link them into Gambit, but I've run out of time. Because of the special form of the Gauss-Lobatto orthogonal polynomials, you can use `sqrt` and `quadratic-solver` to find (by hand) the zeros of $P_5$, which, together with the two endpoints, gives you a 7-point integration rule that's exact for all polynomials of degree $2 \times 7 - 3 = 11$. That's good enough for now.

To repeat what was written in the first part:

The Gauss-Lobatto quadrature rules with $n$ points have the form

$$\int_{-1}^{1} f(x)\,dx \approx \frac{2}{n(n-1)}[f(1)+f(-1)] + \sum_{\nu=0}^{n-3} \gamma_{n\nu} f(x_{n\nu}).$$

Here $x_{n\nu}$ are the zeros of the degree $n-2$ orthogonal polynomial over $[-1,1]$ with the weight

$$w(x) = 1 - x^2.$$

If we define

$$\ell_{n\kappa}(x) = \prod_{\substack{\nu=0\\ \nu\neq\kappa}}^{n} \frac{x - x_{n\nu}}{x_{n\kappa} - x_{n\nu}}$$

then $\ell_{n\kappa}$ has degree $n-1$ and satisfies

$$\ell_{n\kappa}(x_{n\nu}) = \begin{cases} 1, & \nu = \kappa, \\ 0, & \nu \neq \kappa. \end{cases}$$

The weights $\gamma_{n\nu}$ satisfy

$$\gamma_{n\nu} = \int_{-1}^{1} \ell_{n,\nu}(x)\,dx.$$

So now we have all the pieces to find the integration points and weights for a serious numerical integration scheme, which we will use below.