

[Help](#)

```

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2012+2) //The "#els

#else
#include <stdlib.h>
#include "pnl/pnl_vector.h"
#include "pnl/pnl_matrix.h"
#include "pnl/pnl_complex.h"
#include "pnl/pnl_mathtools.h"
#include "pnl/pnl_fft.h"

//-----
void gauleg_pn(double x1, double x2, PnlVect *x, PnlVect *w, int n)
{
    //-----
    // Gauss-Legendre Quadrature Nodes and Weights
    //-----

    int m, j, i;
    double z1, z, xm, x1, pp, p3, p2, p1;
    double EPS_FMM = 3.0e-11;

    m = pnl_iround((n + 1) / 2);
    xm = 0.5 * (x2 + x1);
    x1 = 0.5 * (x2 - x1);
    for (i = 0; i < m; i++)
    {
        z = cos(M_PI * ((i + 1) - 0.25) / (n + 0.5));
        do
        {
            p1 = 1.0;
            p2 = 0.0;
            for (j = 1; j <= n; j++)
            {
                p3 = p2;
                p2 = p1;
                p1 = ((2.0 * j - 1.0) * z * p2 - (j - 1.0) * p3) / j;
            }
            pp = n * (z * p1 - p2) / (z * z - 1.0);
            z1 = z;

```

```

        z = z1 - p1 / pp;
    }
    while (ABS(z - z1) > EPS_FMM);
    pnl_vect_set(x, i, xm - x1 * z);
    pnl_vect_set(w, i, 2.0 * x1 / ((1.0 - z * z)*pp * pp));
    pnl_vect_set(w, n - i - 1, pnl_vect_get(w, i));
    pnl_vect_set(x, n - i - 1, xm + x1 * z);
}
}

```

```

//-----
double interp_lin(double xi, long n, int *start, PnlVect *x, PnlVect *f)
{

```

```

    //-----
    // Linear Interpolation: interpolate from the *start node
    //-----

```

```

    double result = 0., x1, y1, x2, y2;
    int i;

```

```

    for (i = *start + 1; i <= n - 1; i++)
    {

```

```

        x1 = pnl_vect_get(x, i);
        if (x1 >= xi)
        {
            y1 = pnl_vect_get(f, i);
            x2 = pnl_vect_get(x, i - 1);
            y2 = pnl_vect_get(f, i - 1);
            result = (y2 - y1) * (xi - x1) / (x2 - x1) + y1;
            *start = i - 1;
            break;
        }
    }
}

```

```

    return result;
}

```

```

//-----
double interp_lin1(double xi, int n, PnlVect *x, PnlVect *f)
{

```

```

    //-----
    // Linear Interpolation

```

```

//-----
double result, x1, y1, x2, y2;
int i;

result = 0;
for (i = n - 2; i >= 0; i--)
{
    x1 = pnl_vect_get(x, i);
    if (xi >= x1)
    {
        y1 = pnl_vect_get(f, i);
        x2 = pnl_vect_get(x, i + 1);
        y2 = pnl_vect_get(f, i + 1);
        result = (y2 - y1) * (xi - x1) / (x2 - x1) + y1;
        break;
    }
}
return result;
}
//-----
void bmat_mult_vect(PnlMat *K, PnlVect *x, PnlVect *y, int N, int M)
{
    //-----
    // Matrix-Vector Multiplication
    //-----
    // The matrix K is stored as a band matrix: for each column the first(second)
    // the index of the first(last) element different from zero that is stored in
    //-----
    int i, start, len;
    double temp;
    PnlVect *tempK, *temp1, *temp2;

    tempK = pnl_vect_create(M);
    for (i = 0; i < N; i++)
    {
        pnl_mat_get_row(tempK, K, i);
        start = (int)pnl_vect_get(tempK, 0);
        len = (int)pnl_vect_get(tempK, 1);
        temp1 = pnl_vect_create_subvect(tempK, 2, len);
        temp2 = pnl_vect_create_subvect(x, start, len);
        temp = pnl_vect_scalar_prod(temp1, temp2);
    }
}

```

```
        pnl_vect_set(y, i, temp);
        pnl_vect_free(&temp1);
        pnl_vect_free(&temp2);
    }
    pnl_vect_free(&tempK);
}
//-----
#endif //PremiaCurrentVersion
```