1. [Kincaid and Cheney Problem 5.2#1] Find the Schur factorization of

$$A_1 = \begin{bmatrix} 3 & 8 \\ -2 & 3 \end{bmatrix} \quad \text{and} \quad A_2 = \begin{bmatrix} 4 & 7 \\ -1 & 2 \end{bmatrix}.$$

This problem can be worked by hand. For accuracy we use Maple to perform each step in the proof on Schur's theorem. The Maple script is

```
# Kincaid and Cheney Problem 5.2#1
# Schur's factorization for 2x2 matrix

# Written December 2012 by Eric Olson for Math 701

restart;
kernelopts(printbytes=false);
with(LinearAlgebra):
A:=Matrix(A0);
Lambda,X:=Eigenvectors(A);
lambda:=Lambda[1];
x:=X[1..2,1];
x:=x/Norm(x,2);
e1:=Vector([1,0]);
beta:=x[1]/abs(x[1]);
y:=beta*e1;
alpha:=simplify(rationalize(sqrt(2)/Norm(x-y,2)));
v:=simplify(alpha*(x-y));
U:=simplify(IdentityMatrix(2)-v.HermitianTranspose(v));
T:=simplify(U.A.HermitianTranspose(U));
U:=evalf(U);
T:=evalf(T);

matprint:=proc(outfile,M)
    local fd,i,j;
    fd:=open(outfile,WRITE);
    for i from 1 to 2 do
        for j from 1 to 2 do
            if(type(M[i,j],realcons)) then
                fprintf(fd,"%g",M[i,j])
            elif(type(M[i,j],imaginary)) then
                fprintf(fd,"%gI",Im(M[i,j]))
            else
                fprintf(fd,"%Zg",M[i,j])
            fi;
            if(j<2) then
                fprintf(fd,"&")
            else
                fprintf(fd,"\\cr\n")
            fi
        od
    od;
    close(fd)
end proc;

matprint("U.tex",U);
matprint("T.tex",T);
```

This script is run with the commands

```
maple -DA0="[[3,8],[-2,3]]" schur.mpl >schurA1.out
maple -DA0="[[4,7],[1,12]]" schur.mpl >schurA2.out
```

The output indicates that A_1 has a factorization where

$$U_1 \approx \begin{bmatrix} 0.894427 & -0.447214i \\ 0.447214i & -0.894427 \end{bmatrix}$$

and

$$T_1 \approx \begin{bmatrix} 3+4i & -6 \\ 0 & 3-4i \end{bmatrix}$$

and that A_2 has a factorization where

$$U_2 \approx \begin{bmatrix} 0.622704 & 0.782457 \\ 0.782457 & -0.622704 \end{bmatrix}$$

and

$$T_2 \approx \begin{bmatrix} 12.7958 & -6 \\ 0 & 3.20417 \end{bmatrix}.$$

On the following pages is the output from Maple.

```
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
# Kincaid and Cheney Problem 5.2#1
# Schur's factorization for 2x2 matrix
>
# Written December 2012 by Eric Olson for Math 701
>
> restart;
> kernelopts(printbytes=false);
                                    true

> with(LinearAlgebra):
> A:=Matrix([[3,8],[-2,3]]);
                                  [ 3    8]
                              A := [       ]
                                  [-2    3]

> Lambda,X:=Eigenvectors(A);
                                 [3 + 4 I]  [-2 I    2 I]
                   Lambda, X := [       ], [           ]
                                 [3 - 4 I]  [ 1       1 ]

> lambda:=Lambda[1];
                             lambda := 3 + 4 I

> x:=X[1..2,1];
                                   [-2 I]
                              x := [    ]
                                   [ 1  ]

> x:=x/Norm(x,2);
                                   [        1/2]
                                   [-2/5 I 5   ]
                                   [           ]
                             x := [    1/2    ]
                                   [   5       ]
                                   [   ----    ]
                                   [    5      ]

> e1:=Vector([1,0]);
                                     [1]
                                e1 := [ ]
                                     [0]

> beta:=x[1]/abs(x[1]);
                                beta := -I

> y:=beta*e1;
                                     [-I]
                                y := [  ]
                                     [0 ]
```

```
> alpha:=simplify(rationalize(sqrt(2)/Norm(x-y,2)));
                           1/2           1/2 1/2         1/2
                          2    (50 - 20 5   )    (5 + 2 5   )
                alpha := -----------------------------------
                                       10

> v:=simplify(alpha*(x-y));
                     [             1/2           1/2 1/2      ]
                     [     1/10 I 2    (50 - 20 5   )         ]
                     [                                        ]
                v := [ 1/2           1/2 1/2         1/2   1/2]
                     [2    (50 - 20 5   )    (5 + 2 5   ) 5   ]
                     [----------------------------------------]
                     [                  50                    ]

> U:=simplify(IdentityMatrix(2)-v.HermitianTranspose(v));
                            [     1/2                ]
                            [  2 5                 1/2]
                            [  ------      -1/5 I 5   ]
                            [    5                     ]
                       U := [                         ]
                            [                     1/2  ]
                            [       1/2       2 5      ]
                            [1/5 I 5        - ------   ]
                            [                   5      ]

> T:=simplify(U.A.HermitianTranspose(U));
                                [3 + 4 I      -6   ]
                           T := [                   ]
                                [   0       3 - 4 I]

> U:=evalf(U);
                          [ 0.8944271908     -0.4472135954 I]
                     U := [                                ]
                          [0.4472135954 I     -0.8944271908 ]

> T:=evalf(T);
                               [3. + 4. I       -6.   ]
                          T := [                      ]
                               [   0.        3. - 4. I]

>
> matprint:=proc(outfile,M)
>     local fd,i,j;
>     fd:=open(outfile,WRITE);
>     for i from 1 to 2 do
>         for j from 1 to 2 do
>             if(type(M[i,j],realcons)) then
>                 fprintf(fd,"%g",M[i,j])
>             elif(type(M[i,j],imaginary)) then
>                 fprintf(fd,"%gI",Im(M[i,j]))
>             else
>                 fprintf(fd,"%Zg",M[i,j])
>             fi;
>             if(j<2) then
```

```
>                 fprintf(fd,"&")
>             else
>                 fprintf(fd,"\\cr\n")
>             fi
>         od
>     od;
>     close(fd)
> end proc;
matprint := proc(outfile, M)
local fd, i, j;
    fd := open(outfile, WRITE);
    for i to 2 do for j to 2 do
            if type(M[i, j], realcons) then fprintf(fd, "%g", M[i, j])
            elif type(M[i, j], imaginary) then
                fprintf(fd, "%gI", Im(M[i, j]))
            else fprintf(fd, "%Zg", M[i, j])
            end if;
            if j < 2 then fprintf(fd, "&")
            else fprintf(fd, "\cr
                ")
            end if
        end do
    end do;
    close(fd)
end proc

>
> matprint("U.tex",U);
> matprint("T.tex",T);
> quit
bytes used=5311532, alloc=3210676, time=0.73
```

```
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
# Kincaid and Cheney Problem 5.2#1
# Schur's factorization for 2x2 matrix
>
# Written December 2012 by Eric Olson for Math 701
>
> restart;
> kernelopts(printbytes=false);
                                    true

> with(LinearAlgebra):
> A:=Matrix([[4,7],[1,12]]);
                                  [4     7]
                              A := [       ]
                                  [1    12]

> Lambda,X:=Eigenvectors(A);
                           [      1/2]  [    7            7    ]
                           [8 + 23   ]  [---------    ---------]
              Lambda, X := [         ], [      1/2          1/2]
                           [      1/2]  [4 + 23       4 - 23   ]
                           [8 - 23   ]  [                       ]
                                        [    1            1    ]

> lambda:=Lambda[1];
                                        1/2
                            lambda := 8 + 23

> x:=X[1..2,1];
                                  [    7    ]
                                  [---------]
                              x := [      1/2]
                                  [4 + 23   ]
                                  [         ]
                                  [    1    ]

> x:=x/Norm(x,2);
                        [                7                ]
                        [---------------------------------]
                        [/         49     \1/2        1/2 ]
                        [|1 + ------------|    (4 + 23   )]
                        [|           1/2 2|               ]
                        [\    (4 + 23   ) /               ]
                  x := [                                 ]
                        [                1                ]
                        [      ----------------------     ]
                        [      /         49     \1/2      ]
                        [      |1 + ------------|         ]
                        [      |           1/2 2|         ]
                        [      \    (4 + 23   ) /         ]
```

```
> e1:=Vector([1,0]);
                                        [1]
                                  e1 := [ ]
                                        [0]

> beta:=x[1]/abs(x[1]);
                                  beta := 1

> y:=beta*e1;
                                        [1]
                                   y := [ ]
                                        [0]

> alpha:=simplify(rationalize(sqrt(2)/Norm(x-y,2)));
                     1/2 3/4   1/2  1/2   1/2            1/2 1/4   1/2   1/2
          16 (11 + 23   )    %1    2    23      8 (11 + 23   )    %1    23
alpha := - ---------------------------------- - ----------------------------
                          49                                 7

                  1/2 3/4   1/2  1/2        1/2         1/2 1/4
       78 (11 + 23   )    %1    2      39 %1    (11 + 23   )
     + ---------------------------- + ------------------------
                   49                          7

           1/2             1/2 1/2
%1 := -14 2    + 8 (11 + 23   )

> v:=simplify(alpha*(x-y));
v :=

    [             1/2 3/4   1/2  1/2           1/2 3/4   1/2  1/2   1/2]
    [  11 (11 + 23   )    %1    2      (11 + 23   )    %1    2    23   ]
    [- ---------------------------- + ---------------------------------]
    [               392                             392                ]

    [           1/2 3/4   1/2  1/2   1/2           1/2 1/4   1/2   1/2
    [15 (11 + 23   )    %1    2    23      (11 + 23   )    %1    23
    [----------------------------------- + ---------------------------
    [                392                                 7

                  1/2 3/4   1/2  1/2       1/2         1/2 1/4]
       67 (11 + 23   )    %1    2      4 %1    (11 + 23   )   ]
     - ---------------------------- - ------------------------]
                   392                           7           ]

           1/2             1/2 1/2
%1 := -14 2    + 8 (11 + 23   )

> U:=simplify(IdentityMatrix(2)-v.HermitianTranspose(v));
U :=

    [    1/2         1/2 1/2           1/2 1/2  1/2   1/2
    [11 2    (11 + 23   )      (11 + 23   )    2    23
    [---------------------- - -------------------------- ,
    [          56                          56
```

```
          1/2 1/2  1/2   1/2      1/2         1/2 1/2]
    (11 + 23   )    2    23      3 2    (11 + 23   )   ]
    ------------------------- + ----------------------]
               56                          56          ]

    [        1/2 1/2  1/2   1/2      1/2         1/2 1/2
    [(11 + 23   )    2    23      3 2    (11 + 23   )
    [------------------------- + ---------------------- ,
    [            56                          56

          1/2         1/2 1/2           1/2 1/2  1/2   1/2]
      11 2    (11 + 23   )      (11 + 23   )    2    23   ]
    - ---------------------- + --------------------------]
               56                          56            ]
```

```
> T:=simplify(U.A.HermitianTranspose(U));
                                [      1/2             ]
                                [8 + 23          -6    ]
                           T := [                      ]
                                [                   1/2]
                                [    0        8 - 23   ]
```

```
> U:=evalf(U);
                              [0.6227042073    0.7824573279 ]
                         U := [                             ]
                              [0.7824573279    -0.6227042073]
```

```
> T:=evalf(T);
                               [12.79583152        -6.    ]
                          T := [                          ]
                               [    0.         3.204168477]
```

```
>
> matprint:=proc(outfile,M)
>     local fd,i,j;
>     fd:=open(outfile,WRITE);
>     for i from 1 to 2 do
>         for j from 1 to 2 do
>             if(type(M[i,j],realcons)) then
>                 fprintf(fd,"%g",M[i,j])
>             elif(type(M[i,j],imaginary)) then
>                 fprintf(fd,"%gI",Im(M[i,j]))
>             else
>                 fprintf(fd,"%Zg",M[i,j])
>             fi;
>             if(j<2) then
>                 fprintf(fd,"&")
>             else
>                 fprintf(fd,"\\cr\n")
>             fi
>         od
>     od;
>     close(fd)
> end proc;
```

```
matprint := proc(outfile, M)
local fd, i, j;
    fd := open(outfile, WRITE);
    for i to 2 do for j to 2 do
            if type(M[i, j], realcons) then fprintf(fd, "%g", M[i, j])
            elif type(M[i, j], imaginary) then
                fprintf(fd, "%gI", Im(M[i, j]))
            else fprintf(fd, "%Zg", M[i, j])
            end if;
            if j < 2 then fprintf(fd, "&")
            else fprintf(fd, "\cr
                ")
            end if
        end do
    end do;
    close(fd)
end proc

>
> matprint("U.tex",U);
> matprint("T.tex",T);
> quit
bytes used=35862804, alloc=6814496, time=5.74
```

2. [Kincaid and Cheney Problem 5.2#16] Prove that if $(I - vv^*)x = y$ for three vectors v, x and y then $\langle x, y\rangle$ is real.

Computing yields

$$\begin{aligned}\langle x, y\rangle &= \langle x, (I - vv^*)x\rangle = \langle x, x - (\bar{v} \cdot x)v\rangle = \|x\|^2 - (v \cdot \bar{x})\langle x, v\rangle \\ &= \|x\|^2 - \langle v, c\rangle\langle x, v\rangle = \|x\|^2 - |\langle x, v\rangle|^2 \in \mathbf{R}.\end{aligned}$$

3. [Kincaid and Cheney Problem 5.2#34] Let $U = I - \lambda uu^*$ where u is a given vector. Find all the complex values of λ for which U is unitary.

Write $\lambda = a + ib$. Since U is unitary then

$$\begin{aligned} I = UU^* &= (I - \lambda uu^*)(I - \lambda uu^*)^* = (I - \lambda uu^*)(I - \bar{\lambda} uu^*) \\ &= I - (\lambda + \bar{\lambda})uu^* + |\lambda|^2 uu^* uu^* = I - 2auu^* + (a^2 + b^2)|u|^2 uu^* \end{aligned}$$

implies $-2a + (a^2 + b^2)|u|^2 = 0$. Completing the square yields

$$\left(a - \frac{1}{|u|^2}\right)^2 + b^2 = \frac{1}{|u|^4}$$

which is a circle of radius $1/|u|^2$ centered at $1/|u|^2$. Therefore any λ on the circle

$$\left\{ \lambda \in \mathbf{C} : \left| \lambda - \frac{1}{|u|^2} \right| = \frac{1}{|u|^2} \right\}$$

ensures that U is unitary.

4. [Kincaid and Cheney Problem 5.2#39] Prove that $\det(I + xx^*) = 1 + x^*x$.

Let $x \in \mathbf{R}^n$. Define

$$\beta = \|x\| \begin{cases} 1 & \text{if } x_1 = 0 \\ x_1/|x_1| & \text{otherwise} \end{cases}$$

and let $y = \beta e_1$. Then

$$\|y\|^2 = |\beta|^2 = \|x\|^2 \quad \text{and} \quad \langle x, y \rangle = x_1 \bar{\beta} = |x_1| \in \mathbf{R}$$

imply by The Second Lemma on Unitary Matrices from page 267 of Kincaid and Cheney that there exists a unitary matrix U of the form $I - vv^*$ such that $Ux = y$. Now

$$\begin{aligned} \det(I + xx^*) &= \det\left(U(I + xx^*)U^*\right) = \det(UU^* + Uxx^*U^*) \\ &= \det(I + yy^*) = \det(I + |\beta|^2 e_1 e_1^*) = 1 + |\beta|^2 = 1 + x^*x. \end{aligned}$$

5. [Kincaid and Cheney Problem 5.3#7] What is the determinant of a unitary matrix? What is the determinant of an orthogonal matrix?

Let $U \in \mathbf{C}^{n\times n}$ be a unitary matrix. By definition $UU^* = I$. Therefore

$$1 = \det(I) = \det(UU^*) = \det(U)\det(U^*) = \det(U)\overline{\det(U^T)} = |\det(U)|^2.$$

It follows that $\det(U) = e^{i\theta}$ for some $\theta \in \mathbf{R}$. Conversely, suppose $\theta \in \mathbf{R}$. Let $U \in \mathbf{C}^{n\times n}$ be the diagonal matrix such that $u_{11} = e^{i\theta}$ and $u_{jj} = 1$ for $j > 1$. Then $\det U = e^{i\theta}$. Moreover UU^* is the diagonal matrix with diagonal entries $[UU^*]_{jj} = u_{jj}\bar{u}_{jj} = 1$ for $j = 1, \ldots, n$. Thus $UU^* = I$ and so U is unitary.

Now let $O \in \mathbf{R}^{n\times n}$ be an orthogonal matrix. By definition $OO^T = I$. Therefore

$$1 = \det(I) = \det(OO^T) = \det(O)\det(O^T) = \det(O)^2.$$

Since $\det(O) \in \mathbf{R}$ it follows that $\det(O) = \pm 1$. Conversely, given $\alpha = \pm 1$ define $O \in \mathbf{R}^{n\times n}$ to be the diagonal matrix such that $o_{11} = \alpha$ and $o_{jj} = 1$ for $j > 1$. Clearly $OO^T = I$ and so O is orthogonal.

6. [Kincaid and Cheney Problem 5.3#11] A matrix A such that $A^2 = I$ is called an involution or is said to be involutory. Find the necessary and sufficient conditions of u and v^* in order that $I - uv^*$ be an involution.

Let $A \in \mathbf{C}^{n\times n}$ be such that $A^2 = I$. Then

$$\begin{aligned} I = A^2 = (I - uv^*)^2 &= I - 2uv^* + uv^*uv^* \\ &= I - 2uv^* + (\bar{v} \cdot u)uv^* = I - 2uv^* + \langle u, v\rangle uv^* \end{aligned}$$

implies $\langle u, v\rangle = 2$ or that $u = 0$ or $v = 0$. On the other hand, if either $\langle u, v\rangle = 2$ or $u = 0$ or $v = 0$, then the same equality shows that $I - uv^*$ is an involution.

Therefore, it is a necessary and sufficient condition that either $\langle u, v\rangle = 2$ or $u = 0$ or $v = 0$ in order for $I - uv^*$ to be an involution.

7. [Kincaid and Cheney Problem 6.2#4] Prove if f is a polynomial of degree k, then for $n > k$ that $f[x_0, x_1, \ldots, x_n] = 0$.

Let $n > k$ and $p_n(x)$ be the interpolating polynomial of the form

$$p_n(x) = \sum_{i=0}^{n} c_i \prod_{j=0}^{i-1} (x - x_j)$$

passing through the points $\big(j_k, f(x_j)\big)$ where $j = 0, \ldots, n$. By definition

$$c_i = f[x_0, x_1, \ldots, x_i].$$

By the Theorem of Polynomial Interpolation Error from page 315 in Kincaid and Cheney we have that

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{i=0}^{n} (x - x_i).$$

Since $f(x)$ is a polynomial of degree k then $f^{(n+1)}(x) = 0$. Therefore $f(x) = p_n(x)$. This implies that $p_n(x)$ is a polynomial of degree k. Thus $c_n = f[x_0, x_1, \ldots, x_n] = 0$.

8. [Kincaid and Cheney Problem 6.2#25] Establish an iteration method for solving $f(x) = 0$ as follows: Let q_n be the quadratic interpolating polynomial through the points $\big(x_n, f(x_n)\big)$, $\big(x_{n-1}, f(x_{n-1})\big)$ and $\big(x_{n-2}, f(x_{n-2})\big)$ and define x_{n+1} to be the zero of q_n closest to x_n.

As an aside this method was established by D.E. Müller in 1956 and is discussed in detail in *A Survey of Numerical Mathematics* by D. Young and R. Gregory.

Newton's divided difference formula yields

$$q_n(x) = f[x_n] + f[x_n, x_{n-1}](x - x_n) + f[x_n, x_{n-1}, x_{n-2}](x - x_n)(x - x_{n-1})$$

where the $f[x]$, $f[x, y]$ and $f[x, y, z]$ are given recursively as

$$\begin{aligned} f[x] &= f(x) \\ f[x, y] &= (f[y] - f[x])/(y - x) \\ f[x, y, z] &= (f[y, z] - f[x, y])/(z - x). \end{aligned}$$

Writing $c_0 = f[x_n]$, $c_1 = f[x_n, x_{n-1}]$ and $c_2 = f[x_n, x_{n-1}, x_{n-2}]$ we see that

$$\begin{aligned} q_n(x) &= c_0 + \big(c_1 + c_2(x_n - x_{n-1})\big)(x - x_n) + c_2(x - x_n)^2 \\ &= a(x - x_n)^2 + b(x - x_n) + c \end{aligned}$$

where $a = c_2$, $b = c_1 + c_2(x_n - x_{n-1})$ and $c = c_0$. Now use the quadratic formula to solve for x_{n+1}. First define

$$M_1 = -b + \sqrt{b^2 - 4ac} \qquad \text{and} \qquad M_2 = -b - \sqrt{b^2 - 4ac}.$$

Then

$$x_{n+1} = x_n + \begin{cases} 2c/M_2 & \text{if } |M_1| < |M_2| \\ 2c/M_1 & \text{otherwise.} \end{cases}$$

Note that we have written the quadratic formula as

$$\frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

with the square root in the denominator to minimize loss of precision. For example, if $|M_1|$ is small, this means the terms $-b$ and $\sqrt{b^2 - 4ac}$ nearly cancelled. This cancellation causes the value of M_1 to lose precision. Therefore, we compute x_{n+1} using M_2. Similarly, if M_2 is small, we compute x_{n+1} using M_1.

9. [Kincaid and Cheney Problem 6.2#1] For $n = 5, 10$ and 15 find the Newton interpolating polynomial p_n for the function $f(x) = 1/(1 + x^2)$ on the interval $[-5, 5]$. Use equally spaced nodes. In each case, compute $f(x) - p_n(x)$ for 30 equally spaced points in $[-5, 5]$ in order to see the divergence of p_n from f.

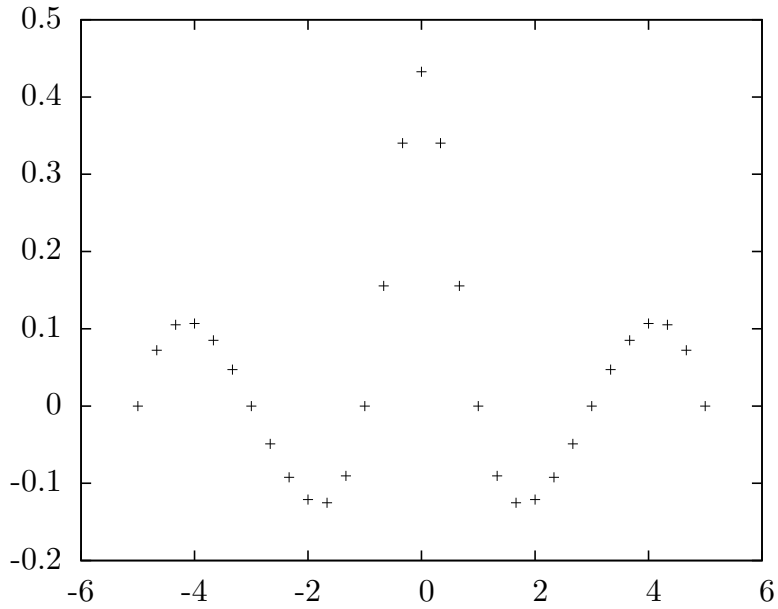
The program is

```
/*
    Kincaid and Cheney Programming Problem 6.2#1
    Polynomial approximation of degree n

    Written December 2012 by Eric Olson for Math 701
*/
#include <stdio.h>
#include <stdlib.h>

void divdif(int n,double x[n+1],double c[n+1]){
    for(int k=1;k<=n;k++) for(int i=n;i>=k;i--) {
        c[i]=(c[i-1]-c[i])/(x[i-k]-x[i]);
    }
}
double interp(int n,double x[n+1],double c[n+1],double z){
    double y=0;
    for(int i=n;i>0;i--) y=(z-x[i-1])*(y+c[i]);
    return y+c[0];
}
double f(double x){
    return 1/(1+x*x);
}

int main(){
    int n;
    scanf("%d",&n);
    printf("# Kincaid and Cheney Programming Problem 6.2#1\n"
        "# Polynomial approximation of degree %d\n\n",n);
    double x[n+1],c[n+1];
    double a=-5,b=5,h=(b-a)/n;
    for(int i=0;i<=n;i++){
        x[i]=a+h*i;
        c[i]=f(x[i]);
    }
    divdif(n,x,c);
    h=(b-a)/30;
    printf("#%17s %18s %18s %18s\n","z","f(z)","Pn(z)","f(z)-Pn(z)");
    for(int i=0;i<=30;i++){
        double z=a+h*i, y=f(z), p=interp(n,x,c,z);
        printf("%18.10e %18.10e %18.10e %18.10e\n",z,y,p,y-p);
    }
    exit(0);
}
```

and my plotting script is

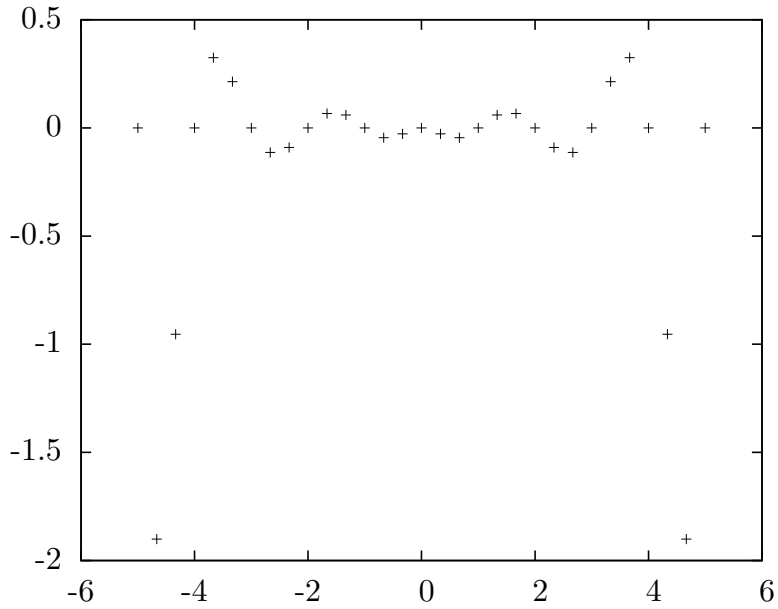
```
set terminal pstex
set key outside
set key width 1
set samples 1000
set size 1.1,0.75
plot datafile using 1:4 ti "$f(x)-p_n(x)$"
```

Output for $n = 5$



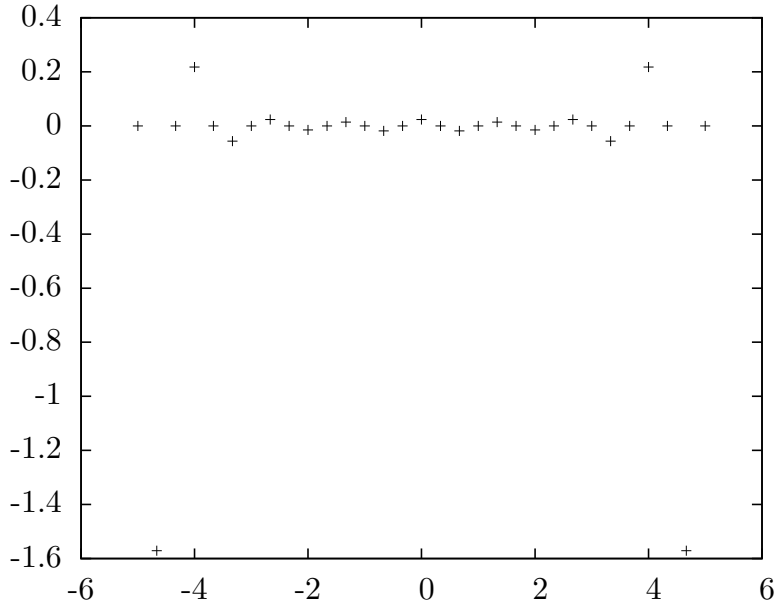

```
# Kincaid and Cheney Programming Problem 6.2#1
# Polynomial approximation of degree 5

#                z              f(z)              Pn(z)          f(z)-Pn(z)
 -5.0000000000e+00   3.8461538462e-02   3.8461538462e-02   0.0000000000e+00
 -4.6666666667e+00   4.3902439024e-02  -2.8323836657e-02   7.2226275682e-02
 -4.3333333333e+00   5.0561797753e-02  -5.4605887939e-02   1.0516768569e-01
 -4.0000000000e+00   5.8823529412e-02  -4.8076923077e-02   1.0690045249e-01
 -3.6666666667e+00   6.9230769231e-02  -1.5859449193e-02   8.5090218424e-02
 -3.3333333333e+00   8.2568807339e-02   3.5493827160e-02   4.7074980179e-02
 -3.0000000000e+00   1.0000000000e-01   1.0000000000e-01   0.0000000000e+00
 -2.6666666667e+00   1.2328767123e-01   1.7224596391e-01  -4.8958292680e-02
 -2.3333333333e+00   1.5517241379e-01   2.4738841406e-01  -9.2216000262e-02
 -2.0000000000e+00   2.0000000000e-01   3.2115384615e-01  -1.2115384615e-01
 -1.6666666667e+00   2.6470588235e-01   3.8983855651e-01  -1.2513267415e-01
 -1.3333333333e+00   3.6000000000e-01   4.5030864198e-01  -9.0308641975e-02
 -1.0000000000e+00   5.0000000000e-01   5.0000000000e-01  -1.1102230246e-16
 -6.6666666667e-01   6.9230769231e-01   5.3691832858e-01   1.5538936372e-01
 -3.3333333333e-01   9.0000000000e-01   5.5963912631e-01   3.4036087369e-01
 -2.7755575616e-16   1.0000000000e+00   5.6730769231e-01   4.3269230769e-01
  3.3333333333e-01   9.0000000000e-01   5.5963912631e-01   3.4036087369e-01
  6.6666666667e-01   6.9230769231e-01   5.3691832858e-01   1.5538936372e-01
  1.0000000000e+00   5.0000000000e-01   5.0000000000e-01   2.2204460493e-16
  1.3333333333e+00   3.6000000000e-01   4.5030864198e-01  -9.0308641975e-02
  1.6666666667e+00   2.6470588235e-01   3.8983855651e-01  -1.2513267415e-01
  2.0000000000e+00   2.0000000000e-01   3.2115384615e-01  -1.2115384615e-01
  2.3333333333e+00   1.5517241379e-01   2.4738841406e-01  -9.2216000262e-02
  2.6666666667e+00   1.2328767123e-01   1.7224596391e-01  -4.8958292680e-02
  3.0000000000e+00   1.0000000000e-01   1.0000000000e-01   2.7755575616e-17
  3.3333333333e+00   8.2568807339e-02   3.5493827160e-02   4.7074980179e-02
  3.6666666667e+00   6.9230769231e-02  -1.5859449193e-02   8.5090218424e-02
  4.0000000000e+00   5.8823529412e-02  -4.8076923077e-02   1.0690045249e-01
  4.3333333333e+00   5.0561797753e-02  -5.4605887939e-02   1.0516768569e-01
  4.6666666667e+00   4.3902439024e-02  -2.8323836657e-02   7.2226275682e-02
  5.0000000000e+00   3.8461538462e-02   3.8461538462e-02   5.3429483060e-16
```

Output for $n = 10$




```
# Kincaid and Cheney Programming Problem 6.2#1
# Polynomial approximation of degree 10

#                z              f(z)              Pn(z)          f(z)-Pn(z)
 -5.0000000000e+00   3.8461538462e-02   3.8461538462e-02   0.0000000000e+00
 -4.6666666667e+00   4.3902439024e-02   1.9446662941e+00   -1.9007638551e+00
 -4.3333333333e+00   5.0561797753e-02   1.0042671440e+00   -9.5370534628e-01
 -4.0000000000e+00   5.8823529412e-02   5.8823529412e-02   0.0000000000e+00
 -3.6666666667e+00   6.9230769231e-02   -2.5560610028e-01   3.2483686951e-01
 -3.3333333333e+00   8.2568807339e-02   -1.3136348377e-01   2.1393229111e-01
 -3.0000000000e+00   1.0000000000e-01   1.0000000000e-01   0.0000000000e+00
 -2.6666666667e+00   1.2328767123e-01   2.3655582000e-01   -1.1326814877e-01
 -2.3333333333e+00   1.5517241379e-01   2.4523053137e-01   -9.0058117580e-02
 -2.0000000000e+00   2.0000000000e-01   2.0000000000e-01   0.0000000000e+00
 -1.6666666667e+00   2.6470588235e-01   1.9862072522e-01   6.6085157131e-02
 -1.3333333333e+00   3.6000000000e-01   3.0030433349e-01   5.9695666510e-02
 -1.0000000000e+00   5.0000000000e-01   5.0000000000e-01   0.0000000000e+00
 -6.6666666667e-01   6.9230769231e-01   7.3724559916e-01   -4.4937906849e-02
 -3.3333333333e-01   9.0000000000e-01   9.2749142537e-01   -2.7491425367e-02
 -2.7755575616e-16   1.0000000000e+00   1.0000000000e+00   1.1102230246e-16
  3.3333333333e-01   9.0000000000e-01   9.2749142537e-01   -2.7491425367e-02
  6.6666666667e-01   6.9230769231e-01   7.3724559916e-01   -4.4937906849e-02
  1.0000000000e+00   5.0000000000e-01   5.0000000000e-01   1.1102230246e-16
  1.3333333333e+00   3.6000000000e-01   3.0030433349e-01   5.9695666510e-02
  1.6666666667e+00   2.6470588235e-01   1.9862072522e-01   6.6085157131e-02
  2.0000000000e+00   2.0000000000e-01   2.0000000000e-01   1.0269562978e-15
  2.3333333333e+00   1.5517241379e-01   2.4523053137e-01   -9.0058117580e-02
  2.6666666667e+00   1.2328767123e-01   2.3655582000e-01   -1.1326814877e-01
  3.0000000000e+00   1.0000000000e-01   1.0000000000e-01   2.5118795932e-15
  3.3333333333e+00   8.2568807339e-02   -1.3136348377e-01   2.1393229111e-01
  3.6666666667e+00   6.9230769231e-02   -2.5560610028e-01   3.2483686951e-01
  4.0000000000e+00   5.8823529412e-02   5.8823529412e-02   6.9527716917e-15
  4.3333333333e+00   5.0561797753e-02   1.0042671440e+00   -9.5370534628e-01
  4.6666666667e+00   4.3902439024e-02   1.9446662941e+00   -1.9007638551e+00
  5.0000000000e+00   3.8461538462e-02   3.8461538462e-02   -4.7066517350e-14
```

Output for $n = 15$




```
# Kincaid and Cheney Programming Problem 6.2#1
# Polynomial approximation of degree 15

#                z              f(z)              Pn(z)          f(z)-Pn(z)
 -5.0000000000e+00   3.8461538462e-02   3.8461538462e-02   0.0000000000e+00
 -4.6666666667e+00   4.3902439024e-02   1.6149814985e+00  -1.5710790595e+00
 -4.3333333333e+00   5.0561797753e-02   5.0561797753e-02   0.0000000000e+00
 -4.0000000000e+00   5.8823529412e-02  -1.5893928816e-01   2.1776281757e-01
 -3.6666666667e+00   6.9230769231e-02   6.9230769231e-02   0.0000000000e+00
 -3.3333333333e+00   8.2568807339e-02   1.3917382108e-01  -5.6605013742e-02
 -3.0000000000e+00   1.0000000000e-01   1.0000000000e-01   0.0000000000e+00
 -2.6666666667e+00   1.2328767123e-01   9.9622123022e-02   2.3665548211e-02
 -2.3333333333e+00   1.5517241379e-01   1.5517241379e-01   0.0000000000e+00
 -2.0000000000e+00   2.0000000000e-01   2.1502247843e-01  -1.5022478430e-02
 -1.6666666667e+00   2.6470588235e-01   2.6470588235e-01   0.0000000000e+00
 -1.3333333333e+00   3.6000000000e-01   3.4583594891e-01   1.4164051091e-02
 -1.0000000000e+00   5.0000000000e-01   5.0000000000e-01   0.0000000000e+00
 -6.6666666667e-01   6.9230769231e-01   7.1094460164e-01  -1.8636909330e-02
 -3.3333333333e-01   9.0000000000e-01   9.0000000000e-01   0.0000000000e+00
 -2.7755575616e-16   1.0000000000e+00   9.7624707634e-01   2.3752923656e-02
  3.3333333333e-01   9.0000000000e-01   9.0000000000e-01   0.0000000000e+00
  6.6666666667e-01   6.9230769231e-01   7.1094460164e-01  -1.8636909330e-02
  1.0000000000e+00   5.0000000000e-01   5.0000000000e-01   4.4408920985e-16
  1.3333333333e+00   3.6000000000e-01   3.4583594891e-01   1.4164051091e-02
  1.6666666667e+00   2.6470588235e-01   2.6470588235e-01   3.0531133177e-15
  2.0000000000e+00   2.0000000000e-01   2.1502247843e-01  -1.5022478430e-02
  2.3333333333e+00   1.5517241379e-01   1.5517241379e-01   7.6882944455e-15
  2.6666666667e+00   1.2328767123e-01   9.9622123022e-02   2.3665548211e-02
  3.0000000000e+00   1.0000000000e-01   1.0000000000e-01   4.6976311729e-14
  3.3333333333e+00   8.2568807339e-02   1.3917382108e-01  -5.6605013742e-02
  3.6666666667e+00   6.9230769231e-02   6.9230769231e-02   8.2156503822e-14
  4.0000000000e+00   5.8823529412e-02  -1.5893928816e-01   2.1776281757e-01
  4.3333333333e+00   5.0561797753e-02   5.0561797753e-02  -4.5050768671e-13
  4.6666666667e+00   4.3902439024e-02   1.6149814985e+00  -1.5710790595e+00
  5.0000000000e+00   3.8461538462e-02   3.8461538464e-02  -2.2520735277e-12
```

10. [Kincaid and Cheney Problem 6.2#2] Program Müller's method for finding roots as described in Problem 6.2.25 above and test it.

The program is

```
/*
    Kincaid and Cheney Programming Problem 6.2#2
    Solve F=0 by Muller's Method

    Written December 2012 by Eric Olson for Math 701
*/
#include <stdio.h>
#include <stdlib.h>
#include <complex.h>
#include <math.h>

#define N 40
#define STR1(s) #s
#define STR2(s) STR1(s)
typedef double complex Complex;
Complex f(Complex z){
    return F;
}
Complex muller(Complex z[3],Complex w[3]){
    Complex c[3];
    for(int k=0;k<3;k++) c[k]=w[k];
    for(int k=1;k<3;k++) for(int i=0;i<3-k;i++)
        c[i]=(c[i+1]-c[i])/(z[i+k]-z[i]);
    Complex b=c[1]+c[0]*(z[2]-z[1]);
    Complex d=csqrt(b*b-4*c[2]*c[0]);
    Complex M1=-b+d,M2=-b-d;
    if(cabs(M1)<cabs(M2)) return z[2]+2*c[2]/M2;
    else return z[2]+2*c[2]/M1;
}
int main(){
    Complex w[N],z[N]={0.25,0.5,0.75};
    printf("# Kincaid and Cheney Programming Problem 6.2#2\n"
         "# Solve %s=0 by Muller's Method\n\n",STR2(F));
    printf("#%2s %17s %17s %17s %17s\n",
        "i","real(z[i])","imag(z[i])","real(w[i])","imag(w[i])");
    for(int i=0;i<3;i++) {
        w[i]=f(z[i]);
        printf("%3d %17.10e %17.10e %17.10e %17.10e\n",
            i,creal(z[i]),cimag(z[i]),creal(w[i]),cimag(w[i]));
    }
    for(int i=3;i<N;i++){
        z[i]=muller(&z[i-3],&w[i-3]);
        w[i]=f(z[i]);
        printf("%3d %17.10e %17.10e %17.10e %17.10e\n",
            i,creal(z[i]),cimag(z[i]),creal(w[i]),cimag(w[i]));
        if(z[i]==z[i-1]) exit(0);
    }
    printf("# Didn't converge!\n");
    exit(0);
}
```

The program can be compiled to solve roots for different functions as

```
gcc -std=gnu99 -DF="z*z+1" -o prog10a prog10.c -lm
gcc -std=gnu99 -DF="cexp(z)+z" -o prog10b prog10.c -lm
gcc -std=gnu99 -DF="z*z*z" -o prog10c prog10.c -lm
```

The output is

```
# Kincaid and Cheney Programming Problem 6.2#2
# Solve z*z+1=0 by Muller's Method

# i        real(z[i])        imag(z[i])        real(w[i])        imag(w[i])
  0  2.5000000000e-01  0.0000000000e+00  1.0625000000e+00  0.0000000000e+00
  1  5.0000000000e-01  0.0000000000e+00  1.2500000000e+00  0.0000000000e+00
  2  7.5000000000e-01  0.0000000000e+00  1.5625000000e+00  0.0000000000e+00
  3  0.0000000000e+00 -1.0000000000e+00  0.0000000000e+00 -0.0000000000e+00
  4  0.0000000000e+00 -1.0000000000e+00  0.0000000000e+00 -0.0000000000e+00
```

and

```
# Kincaid and Cheney Programming Problem 6.2#2
# Solve cexp(z)+z=0 by Muller's Method

# i        real(z[i])        imag(z[i])        real(w[i])        imag(w[i])
  0  2.5000000000e-01  0.0000000000e+00  1.5340254167e+00  0.0000000000e+00
  1  5.0000000000e-01  0.0000000000e+00  2.1487212707e+00  0.0000000000e+00
  2  7.5000000000e-01  0.0000000000e+00  2.8670000166e+00  0.0000000000e+00
  3 -1.1085844094e+00 -7.3847789802e-02 -7.7945808573e-01 -9.8197320828e-02
  4 -4.8978113442e-01  7.9411254946e-03  1.2296003658e-01  1.2807082314e-02
  5 -5.6306863605e-01  9.0034282357e-04  6.3900509273e-03  1.4130510042e-03
  6 -5.6713463412e-01  4.0762660781e-06  1.3565659639e-05  6.3881130461e-06
  7 -5.6714329055e-01 -1.3198551104e-10 -2.1671109351e-10 -2.0684020805e-10
  8 -5.6714329041e-01  4.5536763180e-19  0.0000000000e+00  7.1362632885e-19
  9 -5.6714329041e-01 -3.2710376473e-27  0.0000000000e+00 -5.1261847016e-27
 10 -5.6714329041e-01  0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
 11 -5.6714329041e-01  0.0000000000e+00  0.0000000000e+00  0.0000000000e+00
```

and

```
# Kincaid and Cheney Programming Problem 6.2#2
# Solve z*z*z=0 by Muller's Method

# i        real(z[i])        imag(z[i])        real(w[i])        imag(w[i])
  0  2.5000000000e-01  0.0000000000e+00  1.5625000000e-02  0.0000000000e+00
  1  5.0000000000e-01  0.0000000000e+00  1.2500000000e-01  0.0000000000e+00
  2  7.5000000000e-01  0.0000000000e+00  4.2187500000e-01  0.0000000000e+00
  3  2.2916666667e-01 -9.9913156736e-02  5.1721643519e-03 -1.4744129032e-02
  4  1.8242733623e-01 -1.3330418665e-01 -3.6540716340e-03 -1.0940166999e-02
  5  1.1320131803e-01 -1.3486746718e-01 -4.7265130502e-03 -2.7316571328e-03
  6  4.6023561609e-02 -1.1245265817e-01 -1.6485010517e-03  7.0745042963e-04
  7  1.1607687387e-02 -9.1451368313e-02 -2.8967355195e-04  7.2787401925e-04
  8 -9.9103148828e-03 -6.7765499686e-02  1.3555600737e-04  2.9122358509e-04
  9 -2.0770500757e-02 -4.5953326202e-02  1.2262303100e-04  3.7565331570e-05
 10 -2.4036155562e-02 -2.8722430507e-02  4.5601328675e-05 -2.6086639360e-05
 11 -2.2906696118e-02 -1.5664398875e-02  4.8425545711e-06 -2.0814489640e-05
 12 -1.9475254483e-02 -6.5590742174e-03 -4.8731210290e-06 -7.1811050500e-06
 13 -1.5209134709e-02 -7.7341135865e-04 -3.4908505335e-06 -5.3624876460e-07
 14 -1.1002140304e-02  2.5012212938e-03 -1.1252853490e-06  8.9264877630e-07
 15 -7.3440644873e-03  3.9925003511e-03 -4.4909733064e-08  5.8236921990e-07
 16 -4.4365844552e-03  4.3213996534e-03  1.6122638091e-07  1.7447802315e-07
 17 -2.2967330808e-03  3.9724821450e-03  9.6616349733e-08  1.7611658246e-10
 18 -8.4064006431e-04  3.2941275908e-03  2.6771993616e-08 -2.8761849654e-08
```

```
 19  6.1640117010e-05  2.5180731351e-03 -1.1722888466e-09 -1.5937624687e-08
 20  5.4938226284e-04  1.7834926724e-03 -5.0766862725e-09 -4.0581298001e-09
 21  7.4979191331e-04  1.1616988086e-03 -2.6141078567e-09  3.9151528855e-10
 22  7.6791662094e-04  6.7801675860e-04 -6.0621199341e-10  8.8778231918e-10
 23  6.8376017065e-04  3.2942105091e-04  9.7075677322e-11  4.2629247760e-10
 24  5.5377150915e-04  9.7697631033e-05  1.5396420833e-10  8.8948204981e-11
 25  4.1455403644e-04 -4.1474291277e-05  6.9103960168e-11 -2.1311358469e-11
 26  2.8733132646e-04 -1.1278707803e-04  1.2756509906e-11 -2.6500107687e-11
 27  1.8228363969e-04 -1.3811091152e-04 -4.3741778071e-12 -1.1132754862e-11
 28  1.0228172709e-04 -1.3508677373e-04 -4.5294192110e-12 -1.7745293966e-12
 29  4.5832133906e-05 -1.1687225098e-04 -1.7818059998e-12  8.5987234105e-13
 30  9.2287945586e-06 -9.2551441047e-05 -2.3636915123e-13  7.6912628792e-13
 31 -1.1994753859e-05 -6.7859165502e-05  1.6397698034e-13  2.8319285408e-13
 32 -2.2175025194e-05 -4.5988948670e-05  1.2979526675e-13  2.9423185432e-14
 33 -2.5045773302e-05 -2.8347969344e-05  4.4669921557e-14 -3.0566620755e-14
 34 -2.3552319884e-05 -1.5190386228e-05  3.2391912834e-15 -2.1773709465e-14
 35 -1.9846253763e-05 -6.1062869208e-06 -5.5969128459e-15 -6.9876355555e-15
 36 -1.5379790863e-05 -3.7146558373e-07 -3.6315378443e-15 -2.6354588463e-16
 37 -1.1043277129e-05  2.8220586809e-06 -1.0829250043e-15  1.0100088588e-15
 38 -7.3095232165e-06  4.2292375149e-06  1.6828000373e-18  6.0224739337e-16
 39 -4.3642015246e-06  4.4833962082e-06  1.8005067527e-16  1.6605568481e-16
# Didn't converge!
```

Müller's method quickly finds the roots in the first two cases. In the last case, however, the method converges slowly because the derivative vanishes at the root.