THE TI-NSPIRE PROGRAMS

JAMES KEESLING

The purpose of this document is to list and document the programs that will be used in this class. For each program there is a screen shot containing an example and a listing of the TN-Nspire CX CAS program. The student is responsible to enter each program and be familiar with its use.

1. Solving $f(x) = 0$

In this section are the Newton-Raphson method and the Bisection method.

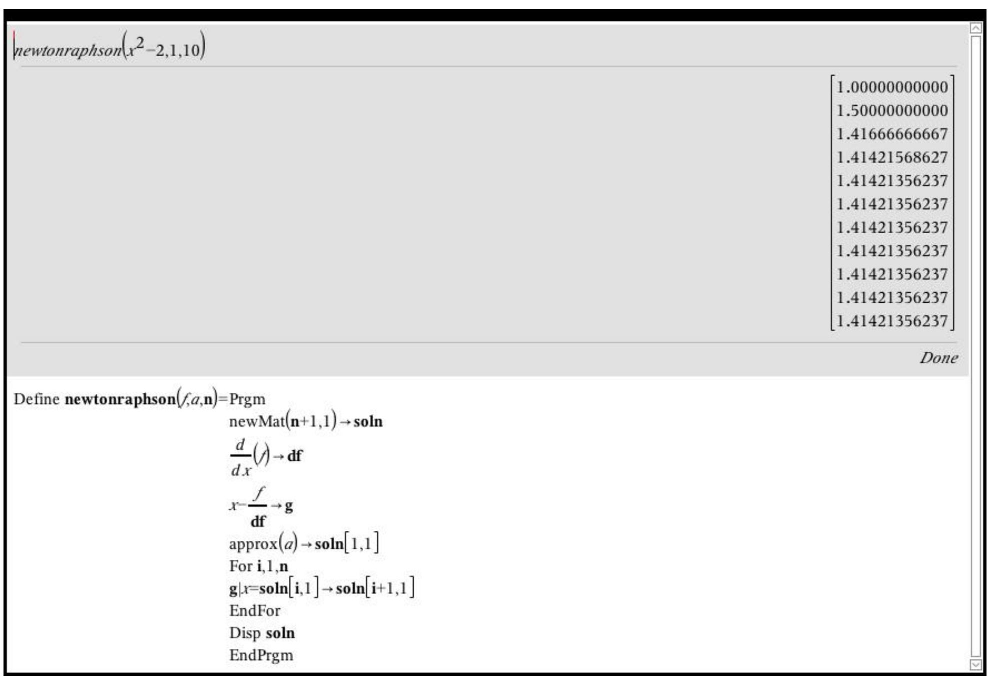


Figure 1. The Newton-Raphson method applied to $x^2 - 2 = 0$

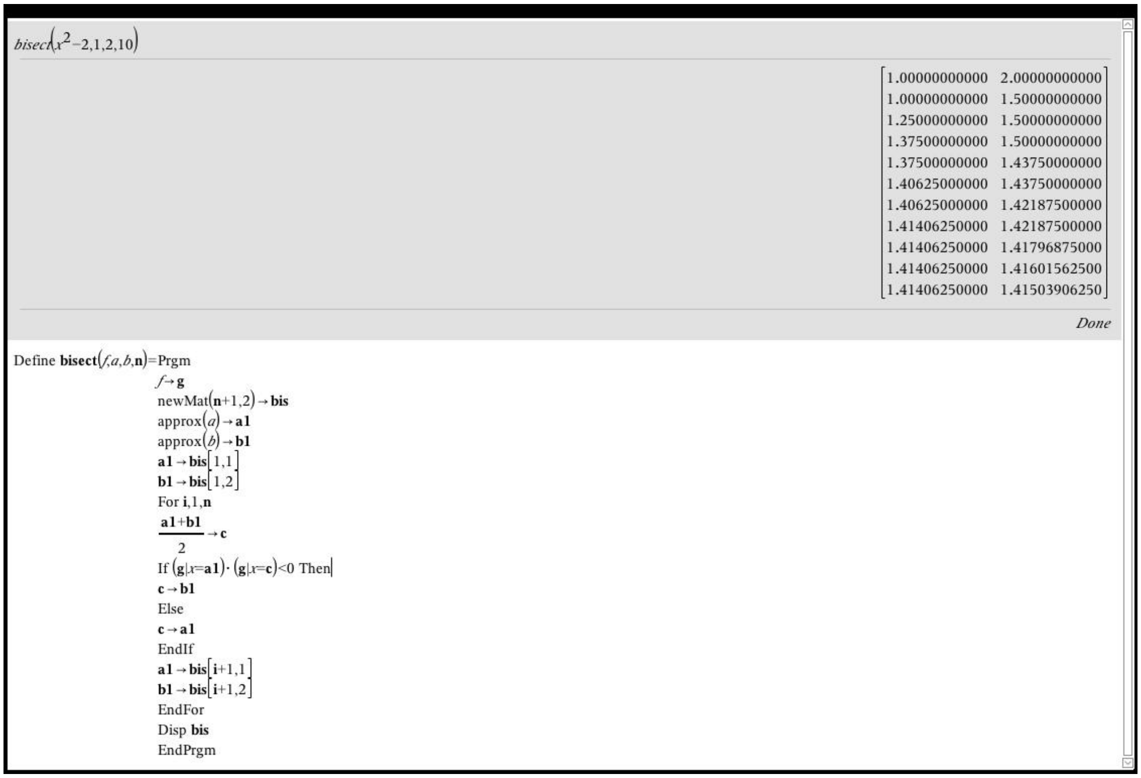


Figure 2. The Bisection method applied to $x^2 - 2 = 0$

2. Lagrange Polynomials

In this section are the programs for the VanderMonde matrix and the Lagrange polynomial through a set of points $\{(x_0, y_0), (x_1, y_1), \ldots (x_n, y_n)\}$. Note that the polynomial program calls for the points in the form $[x_0, x_1, \ldots, x_n]$ and $[y_0, y_1, \ldots, y_n]$.

```
vandermonde([1 2 3])
                                    [1 1 1]
                                    [1 2 4]
                                    [1 3 9]
                                       Done
Define vandermonde(a)=Prgm
    dim(a)[2]-1→n
    newMat(n+1,n+1)→vander
    For i,1,n+1
    For j,1,n+1
    If a[1,i]=0 and j=1 Then
    1→vander[i,j]
    Else
    a[1,i]^(j-1)→vander[i,j]
    EndIf
    EndFor
    EndFor
    Disp vander
    EndPrgm
```

FIGURE 3. Program for the VanderMonde matrix for the points $\{1, 2, 3\}$

```
lagrange([0 1 3],[1 0 2])
                                    [1 0 0]
                                    [1 1 1]
                                    [1 3 9]
                                    2·x^2/3 - 5·x/3 + 1
                                       Done
Define lagrange(a,b)=Prgm
    DelVar x
    dim(a)[2]-1→n
    vandermonde(a)
    newMat(n+1,1)→coef
    vander^-1·b^T→coef
    0→p
    For i,0,n
    p+coef[i+1,1]·x^i→p
    EndFor
    Disp p
    EndPrgm
```

FIGURE 4. The Lagrange polynomial through the points $\{(0, 1), (1, 0), (3, 2)\}$

```
newtonpolynomial([0 1 3],[1 0 2])
```

$$\begin{bmatrix} 1 & -1 & \frac{2}{3} \\ 0 & 1 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

$$\frac{(x-1)\cdot(2\cdot x-3)}{3}$$

Done

```
Define newtonpolynomial(a,b)=Prgm
    dim(a)[2]-1→n
    newMat(n+1,n+1)→coef
    For i,1,n+1
    b[1,i]→coef[i,1]
    EndFor
    For j,1,n
    For i,1,n-j+1
    (coef[i+1,j]-coef[i,j])/(a[1,i+j]-a[1,i])→coef[i,j+1]
    EndFor
    EndFor
    0→p
    For j,1,n+1
    coef[1,j]→temp
    For i,1,j-1
    temp·(x-a[1,i])→temp
    EndFor
    p+temp→p
    EndFor
    Disp coef
    Disp p
    EndPrgm
```

FIGURE 5. The Newton polynomial determined by the points $\{(0,1),(1,0),(3,2)\}$

3. Numerical Integration

In this section are programs to compute the normalized Newton-Cotes coefficients and to estimate an integral using Newton-Cotes integration. We also have programs for Romberg integration and Gaussian integration. These last programs are greatly superior to Newton-Cotes. For Gaussian quadrature we theoretically need the Legendre polynomials. We give a program that will list the first $n+1$ of these polynomials from degree 0 to degree n.

newtoncotes(5)

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 & 32 \\ 1 & 3 & 9 & 27 & 81 & 243 \\ 1 & 4 & 16 & 64 & 256 & 1024 \\ 1 & 5 & 25 & 125 & 625 & 3125 \end{bmatrix}$$

$$\begin{bmatrix} \frac{19}{288} \\ \frac{25}{96} \\ \frac{25}{144} \\ \frac{25}{144} \\ \frac{25}{96} \\ \frac{19}{288} \end{bmatrix}$$

Done

```
Define newtoncotes(n)=Prgm
newMat(n+1,n+1)→vander
newMat(1,n+1)→temp
For i,0,n
i→temp[1,i+1]
EndFor
vandermonde(temp)
For i,0,n
n^(i+1)/(i+1)→temp[1,i+1]
EndFor
(vander^T)^-1·temp^T→coef
1/n·coef→coef
Disp coef
EndPrgm
```

Figure 6. Program for the Newton-Cotes coefficients

romberg(sin(*x*),5,0,π)

$$\begin{bmatrix} 0.00000000000 & 2.09439510239 & 1.99857073182 & 2.00000554998 & 1.99999999459 & 2.00000000000 \\ 1.57079632679 & 2.00455975498 & 1.99998313095 & 2.00000001629 & 2.00000000000 & 0 \\ 1.89611889794 & 2.00026916995 & 1.99999975245 & 2.00000000006 & 0 & 0 \\ 1.97423160195 & 2.00001659105 & 1.99999999619 & 0 & 0 & 0 \\ 1.99357034377 & 2.00000103337 & 0 & 0 & 0 & 0 \\ 1.99839336097 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Done

```
Define romberg(f,n,a,b)=Prgm
f→g
approx(a)→a1
approx(b)→b1
newMat(n+1,n+1)→romb
For i,0,n
(b1-a1)/(2·2^i)·((g|x=a1)+(g|x=b1)+2·Σ(j=1, 2^i-1)(g|x=a1+(j·(b1-a1))/2^i))→romb[i+1,1]
EndFor
For j,1,n
For i,1,n-j+1
(4^j·romb[i+1,j]-romb[i,j])/(4^j-1)→romb[i,j+1]
EndFor
EndFor
Disp romb
EndPrgm
```

Figure 7. Program for Romberg Integration

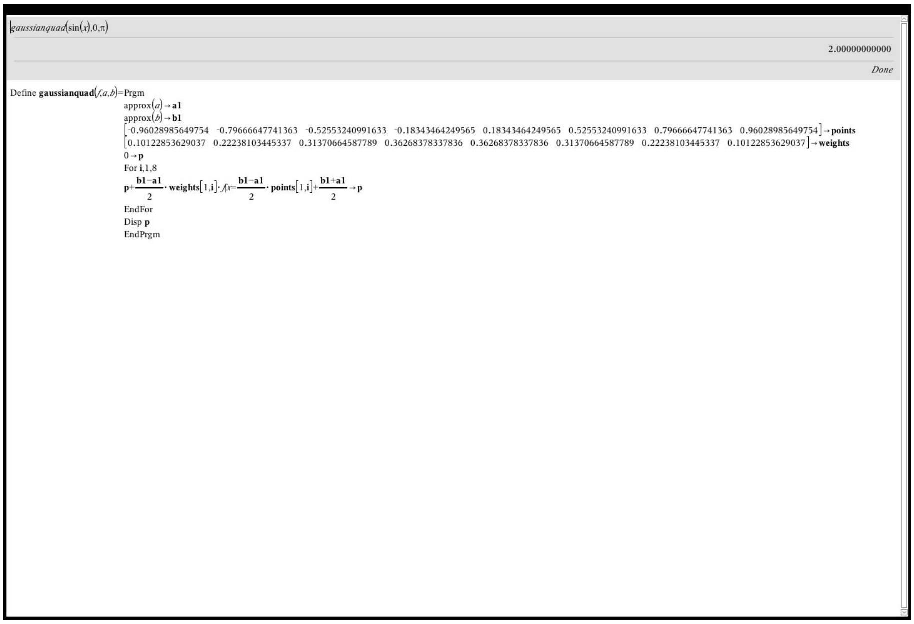


FIGURE 8. Program for Gaussian quadrature

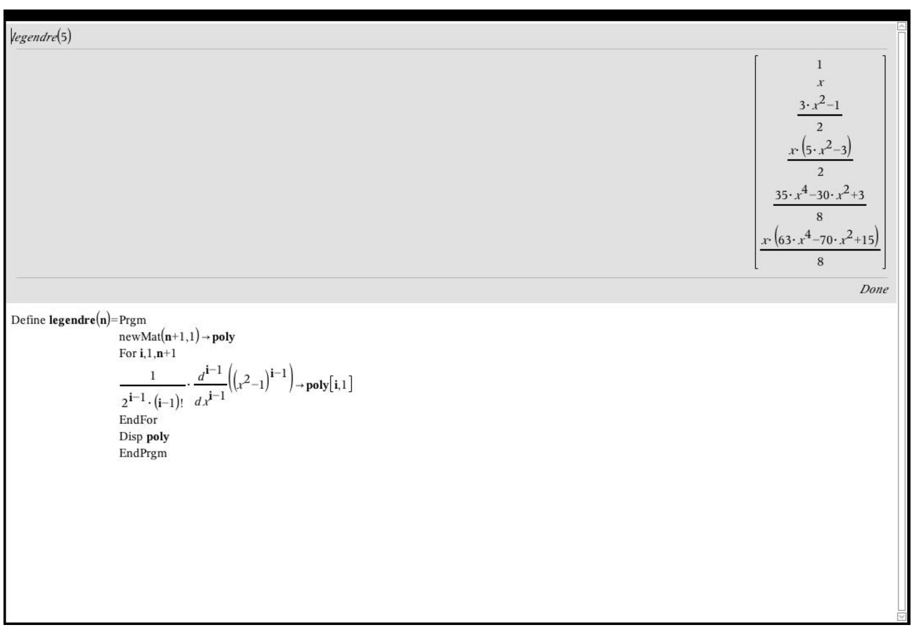
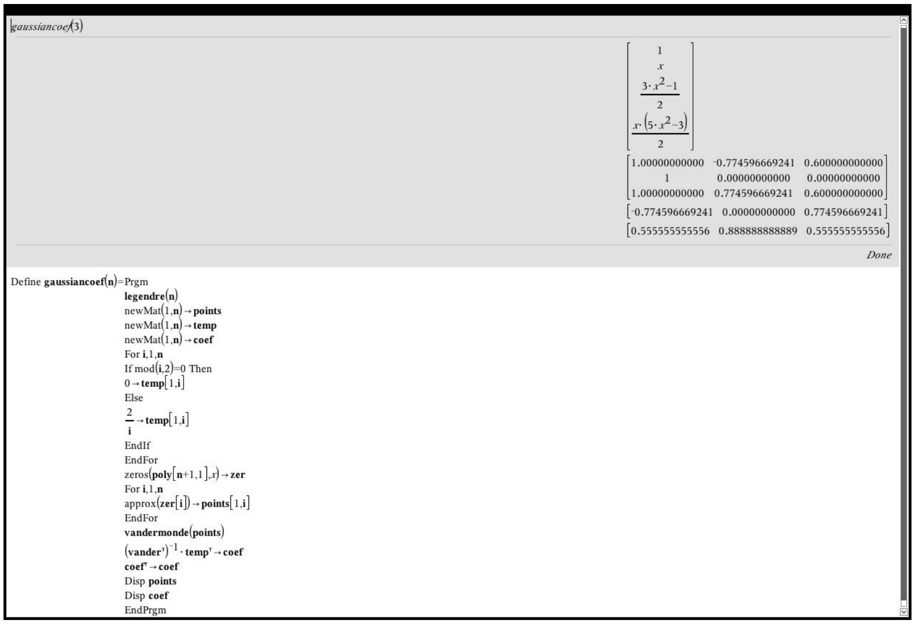


FIGURE 9. Program for the Legendre polynomials up to degree n



```
gaussiancoef(3)

[ 1 ; x ; (3·x^2-1)/2 ; x·(5·x^2-3)/2 ]
[ 1.00000000000   ⁻0.774596669241   0.600000000000
  1               0.00000000000     0.00000000000
  1.00000000000   0.774596669241    0.600000000000 ]
[ ⁻0.774596669241  0.00000000000  0.774596669241 ]
[ 0.555555555556   0.888888888889  0.555555555556 ]
                                             Done

Define gaussiancoef(n)=Prgm
legendre(n)
newMat(1,n)→points
newMat(1,n)→temp
newMat(1,n)→coef
For i,1,n
If mod(i,2)=0 Then
0→temp[1,i]
Else
2/i→temp[1,i]
EndIf
EndFor
zeros(poly[n+1,1],x)→zer
For i,1,n
approx(zer[i])→points[1,i]
EndFor
vandermonde(points)
(vanderᵀ)^-1·tempᵀ→coef
coefᵀ→coef
Disp points
Disp coef
EndPrgm
```

FIGURE 10. Program for n points and n coefficients for Gaussian quadrature

4. Numerical Differentiation

In this section we give the programs needed for numerical differentiation of a function $f(x)$. There are two of these programs. The first program determines the coefficients to be used in estimating $f^{(k)}(a)$ using the $n+1$ points $\{a - m_0 \cdot h, a - m_1 \cdot h, \cdots, a - m_n \cdot h\}$. In the programs $b = [m_0, m_1, \ldots, m_n]$

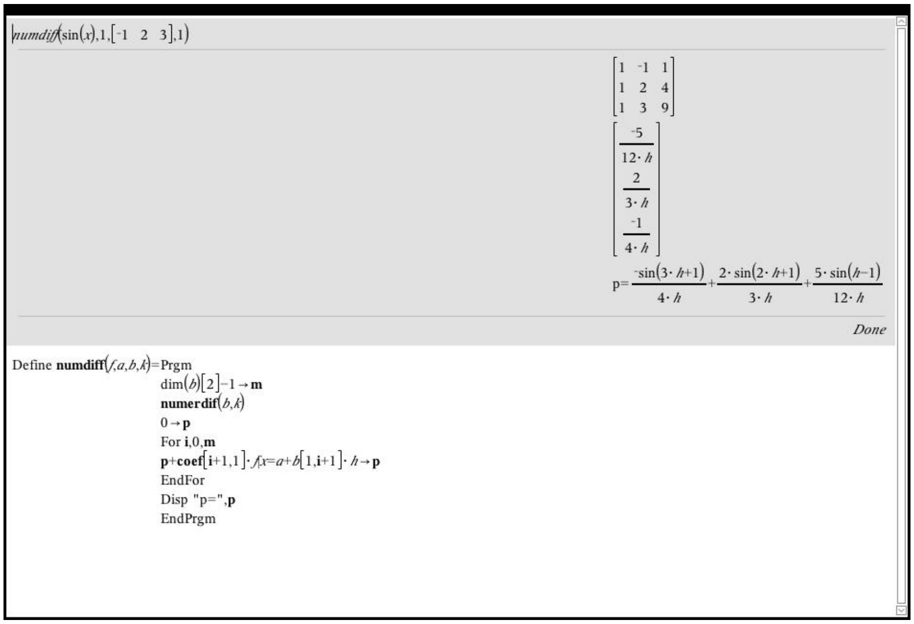


FIGURE 11. Program determining the coefficients to be used in estimating the kth-derivative at a using the points $a - h \cdot b$ with $b = [m_0, m_1, \ldots, m_n]$

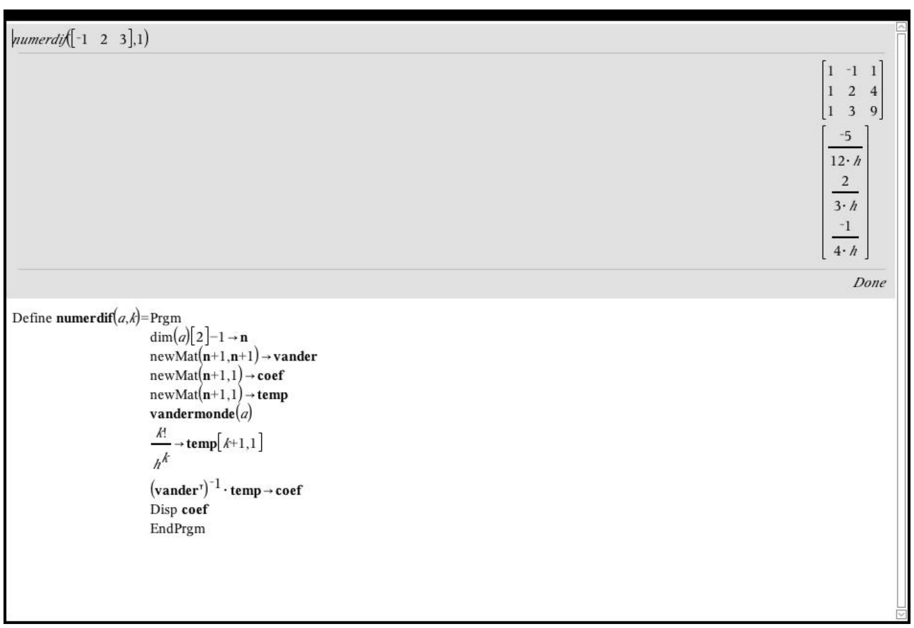


FIGURE 12. Program giving the formula to estimate $f^{(k)}(a)$ using the points $a - h \cdot b$ with $b = [m_0, m_1, \ldots, m_n]$

5. Differential Equations

In this section we give some programs useful for solving ordinary differential equations. We give a theoretical solution based on Picard iteration and numerical methods based some method of integration. We also give a program for the Taylor method.

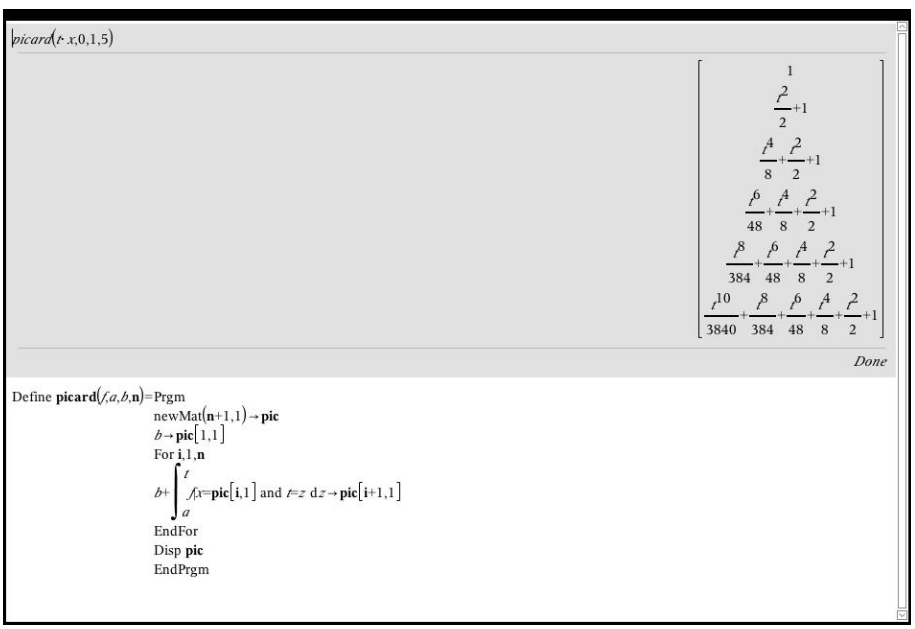


Figure 13. Program giving the Picard iteration method

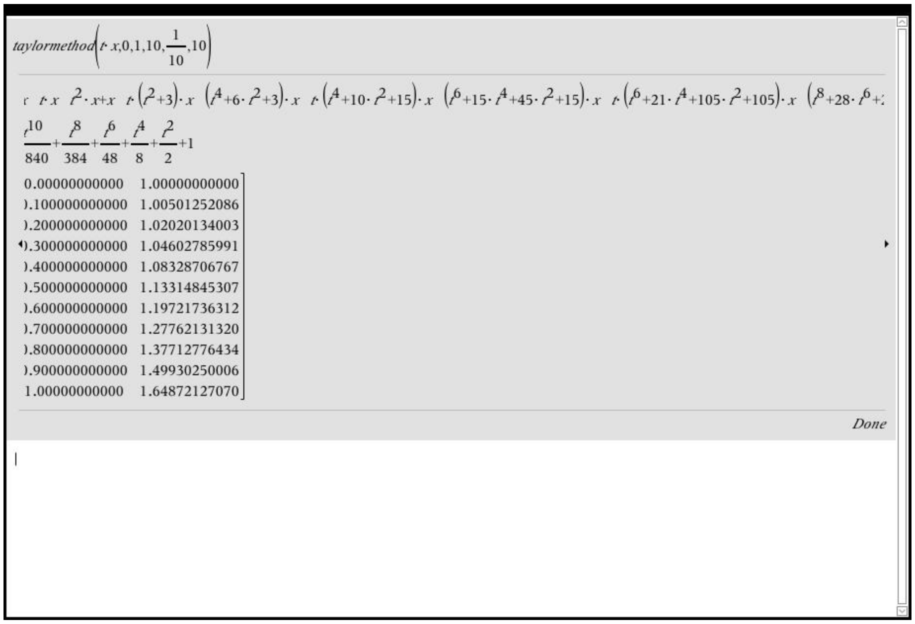


FIGURE 14. Example using the Taylor method

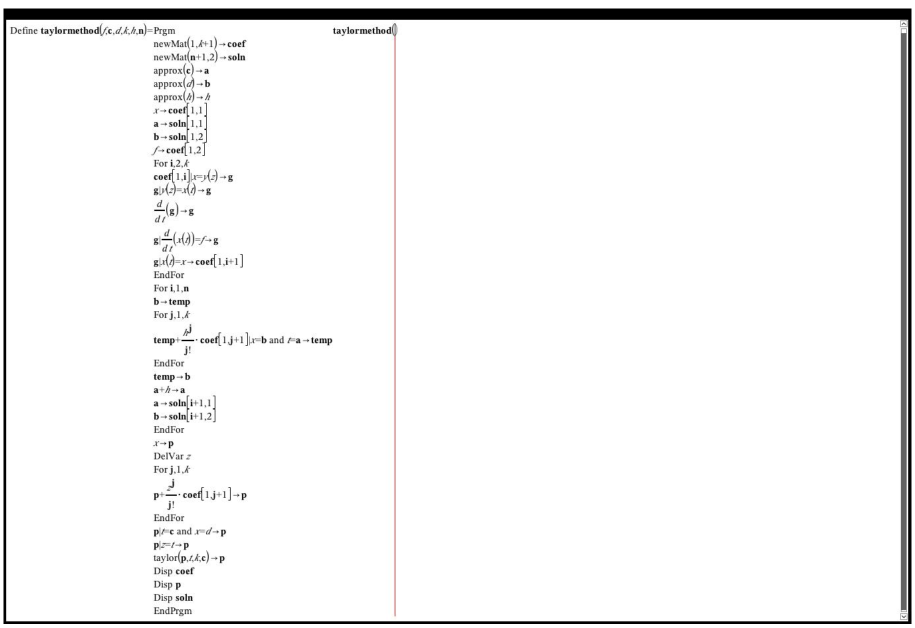


FIGURE 15. Program for the Taylor method

```
eulermethod(t·x,0,1,1/10,10)
```

$$\begin{bmatrix} 0.00000000000 & 1.00000000000 \\ 0.100000000000 & 1.00000000000 \\ 0.200000000000 & 1.01000000000 \\ 0.300000000000 & 1.03020000000 \\ 0.400000000000 & 1.06110600000 \\ 0.500000000000 & 1.10355024000 \\ 0.600000000000 & 1.15872775200 \\ 0.700000000000 & 1.22825141712 \\ 0.800000000000 & 1.31422901632 \\ 0.900000000000 & 1.41936733762 \\ 1.00000000000 & 1.54711039801 \end{bmatrix}$$

Done

```
Define eulermethod(f,a,b,h,n)=Prgm
    newMat(n+1,2)→soln
    approx(a)→a
    approx(b)→b
    approx(h)→h
    a→soln[1,1]
    b→soln[1,2]
    For i,1,n
    b+h·f|t=a and x=b→b
    a+h→a
    a→soln[i+1,1]
    b→soln[i+1,2]
    EndFor
    Disp soln
    EndPrgm
```

Figure 16. Program giving the Euler method

```
modifiedeuler(t·x,0,1,1/10,10)
```

$$\begin{bmatrix} 0.00000000000 & 1.00000000000 \\ 0.100000000000 & 1.00500000000 \\ 0.200000000000 & 1.02015037500 \\ 0.300000000000 & 1.04590917197 \\ 0.400000000000 & 1.08306509530 \\ 0.500000000000 & 1.13277778318 \\ 0.600000000000 & 1.19663813070 \\ 0.700000000000 & 1.27675305355 \\ 0.800000000000 & 1.37586100934 \\ 0.900000000000 & 1.49748712256 \\ 1.00000000000 & 1.64615015665 \end{bmatrix}$$

Done

```
Define modifiedeuler(f,a,b,h,n)=Prgm
    approx(a)→a
    approx(b)→b
    approx(h)→h
    newMat(n+1,2)→soln
    a→soln[1,1]
    b→soln[1,2]
    For i,1,n
    f|x=b and t=a→temp
    b+h·f|t=a+h/2 and x=b+h/2·temp→b
    a+h→a
    a→soln[i+1,1]
    b→soln[i+1,2]
    EndFor
    Disp soln
    EndPrgm
```

Figure 17. Program giving the modified Euler method

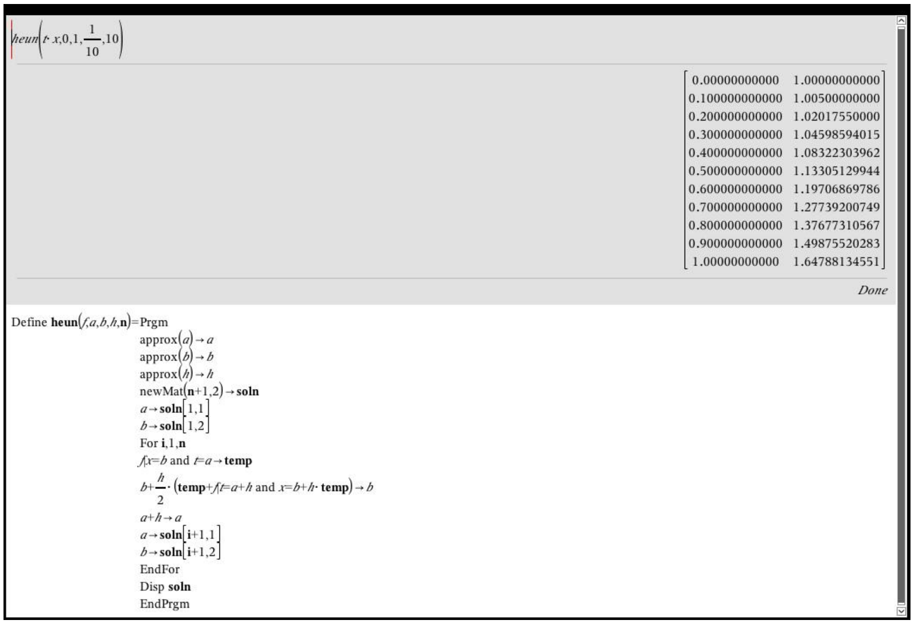


FIGURE 18. Program giving the Heun method

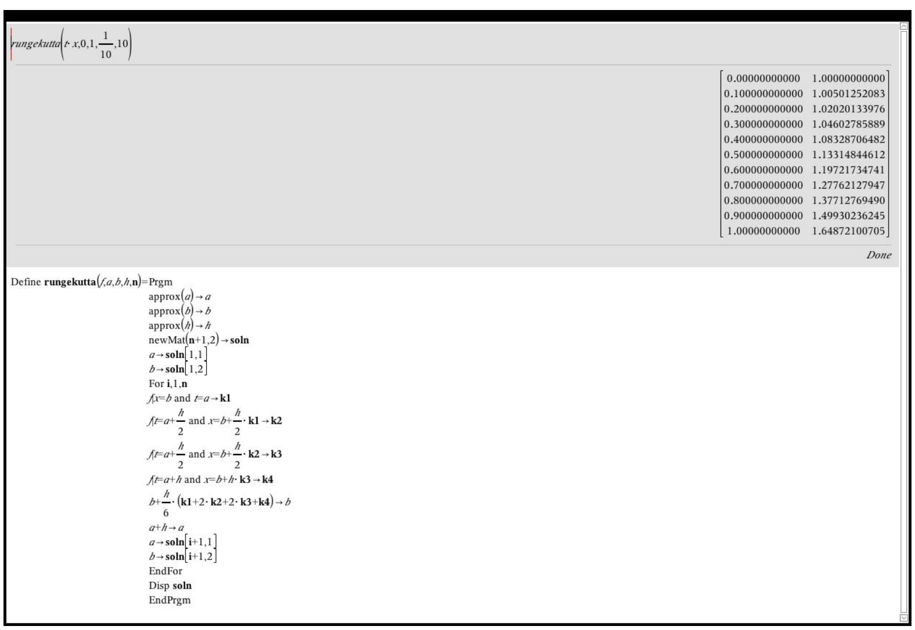


FIGURE 19. Program giving the Runge-Kutta method

6. Stochastic Simulation

In this section we do not show copies of the programs. They can now be downloaded. So, there is no need for them to be given to be copied into your calculator. However, we do give examples of how the data is to be entered when the programs are run and what the typical output will look like and how it should be interpreted.

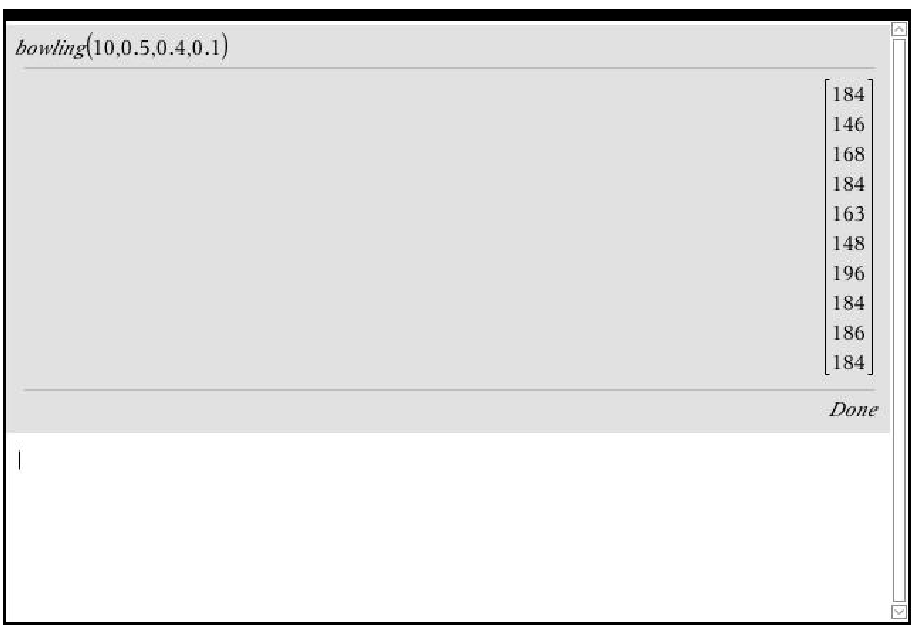


Figure 20. Program giving an example of the Bowling program. The output is a sequence of scores for a bowler whose probability of a strike, spare, and open frame are the numbers entered in that order.

FIGURE 21. Program giving an example of the Queue simulation program. The output gives the time spent in each state given the arrival rate, the service rate, and the number of servers.

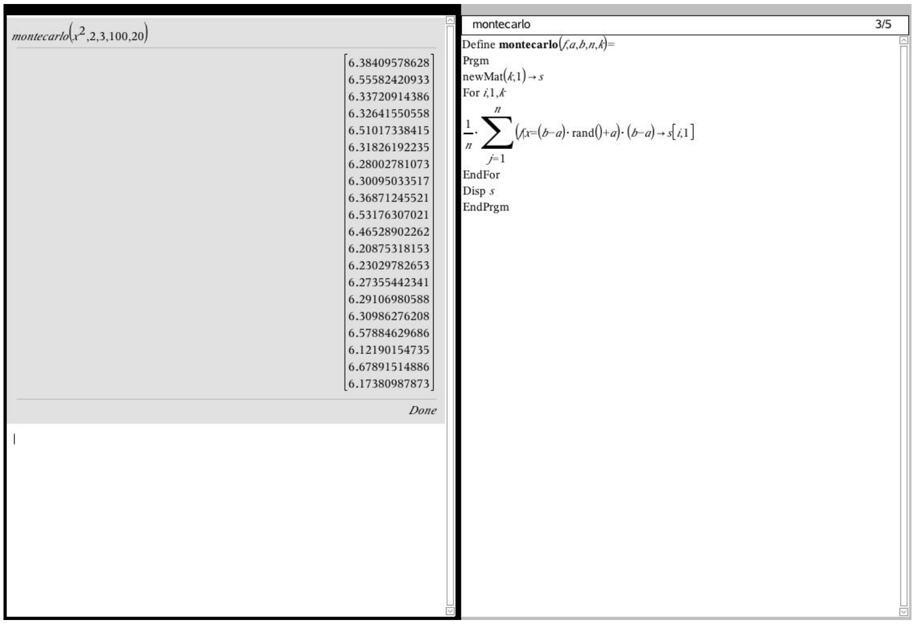


FIGURE 22. This program estimates the integral $\int_a^b dx$ by the Monte Carlo method using n random points in the interval $[a,b]$. It gives k estimates of the integral.