Nonlinear FEM Homework 5

Truman Ellis

For the less nonlinear case of $x = 1.8$, all algorithms are able to correctly approximate the solution, but with different levels of efficiency. With the more nonlinear $x = 2.1$ problem, only the line search algorithms were able to make the jump across the valley and reattain the solution. None of the algorithms were able to approximate the exact solution for cases when $N1$ dipped down again. The algorithms without line search appear to have gotten stalled as the slope approaches zero for the highly nonlinear problem.

It appears that overall, Modified Newton-Rhapson took the most iterations to converge, while adding a BFGS update slightly improved things. Using the consistent tangent for Newton-Rhapson produced even better rates. The line search algorithms outperformed all alternatives and usually converged in one iteration. It appears that the line search algorithms are performing so well because the newton algorithm used to solve for the line search parameter, s, is really driving the residual to zero. It is curious that BFGS is not producing significantly better results than standard Modified Newton-Rhapson, but it does appear to be converging slightly faster. Perhaps the strengths would come out more clearly in a larger or different problem.

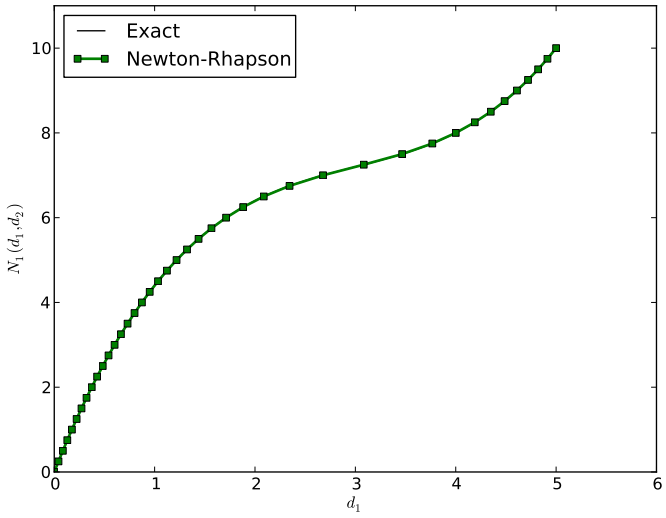

Figure 1: Newton-Rhapson, $x = 1.8$

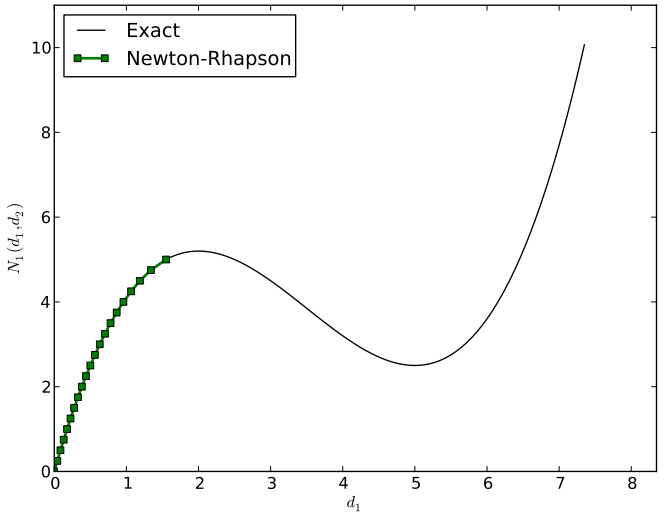

Figure 2: Newton-Rhapson, $x = 2.1$

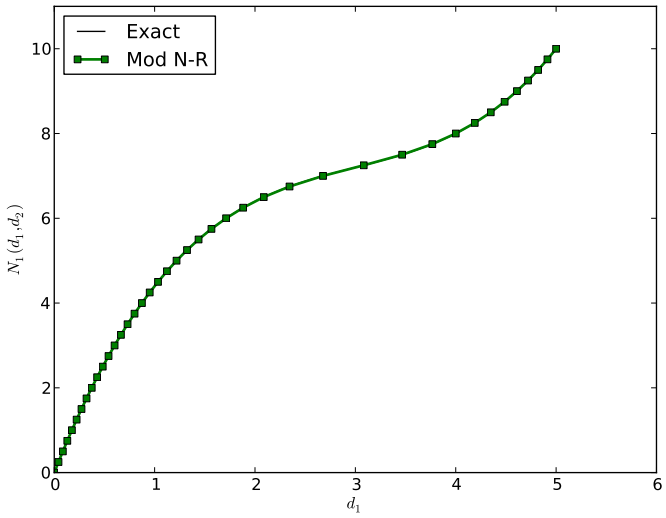

Figure 3: Modified Newton-Rhapson, $x = 1.8$

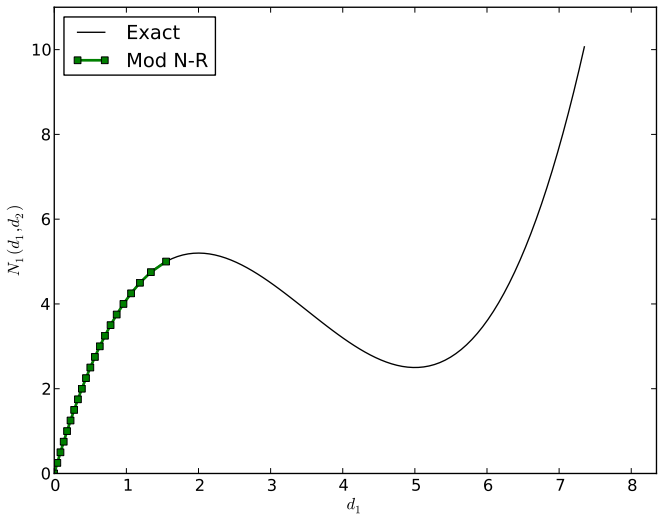

Figure 4: Modified Newton-Rhapson, $x = 2.1$

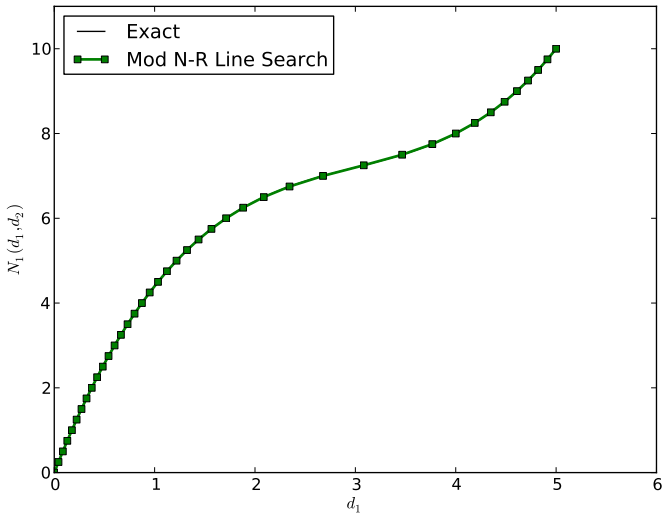

Figure 5: Modified Newton-Rhapson with Line Search, $x = 1.8$

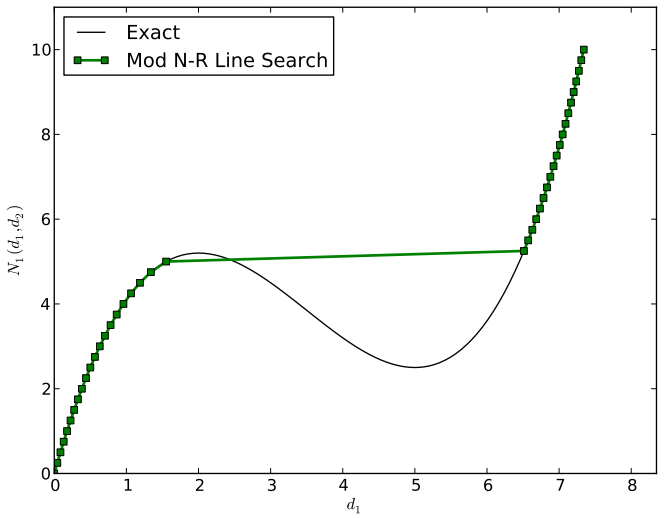

Figure 6: Modified Newton-Rhapson with Line Search, $x = 2.1$

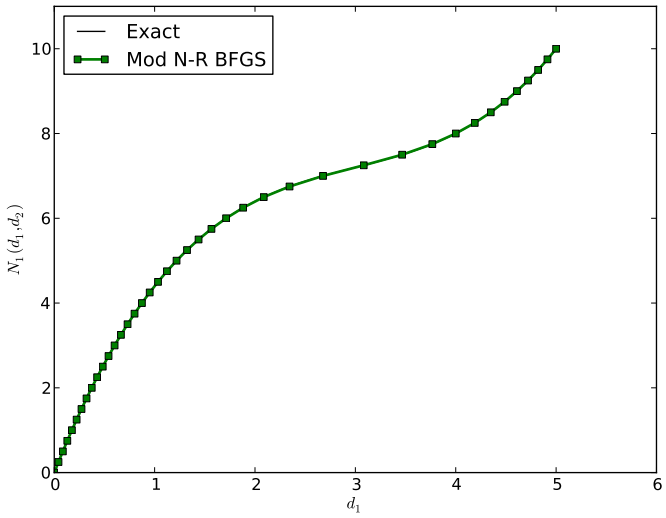

Figure 7: Modified Newton-Rhapson with BFGS, $x = 1.8$

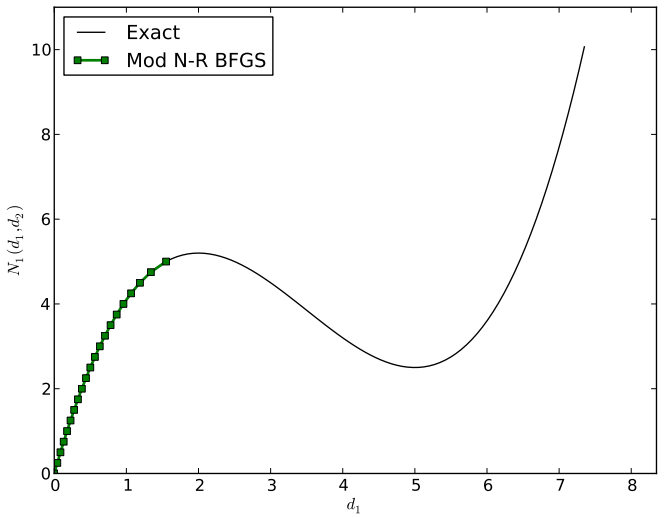

Figure 8: Modified Newton-Rhapson with BFGS, $x = 2.1$

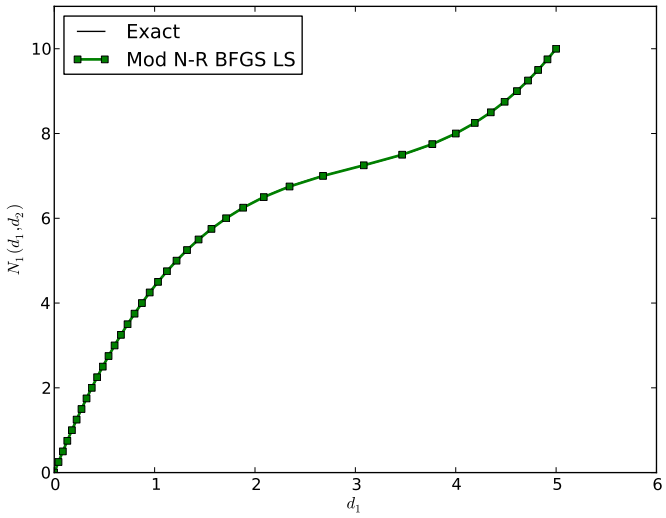

Figure 9: Modified Newton-Rhapson with BFGS and Line Search, $x = 1.8$

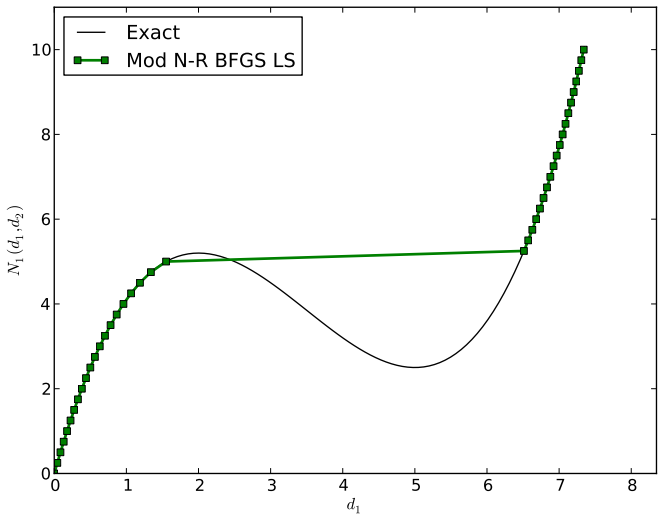

Figure 10: Modified Newton-Rhapson with BFGS and Line Search, $x = 2.1$

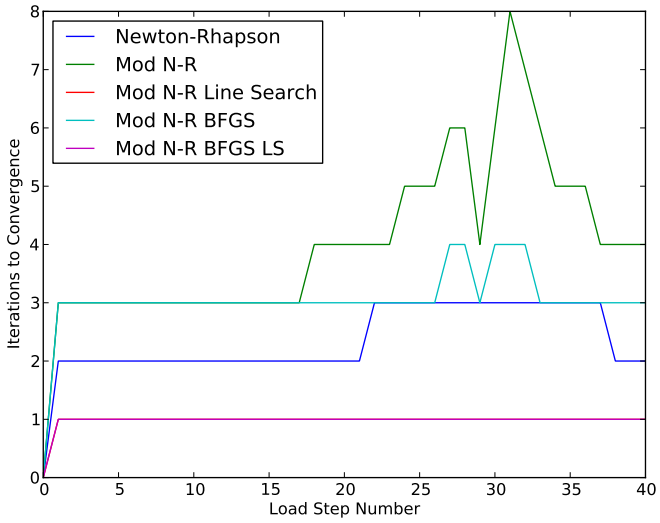

Figure 11: Iterations to convergence, $x = 1.8$

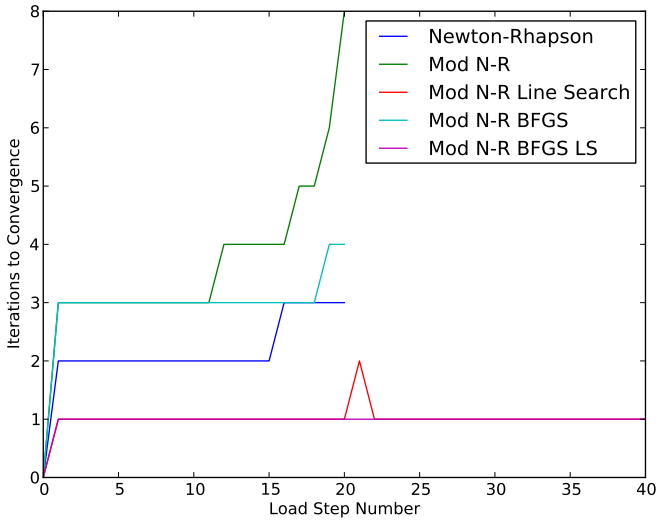

Figure 12: Iterations to convergence, $x = 2.1$

Homework5.py

```
from pylab import *
from scipy import optimize
close('all')

N = 41
#x = 1.8
x = 2.1
epsilon = 1e-4

d1 = 0
if (x == 1.8):
  d1 = linspace(0,5,100)
else:
  d1 = linspace(0,7.35,100)
N1ex = 0.2*d1**3-x*d1**2+6*d1
for i in range(1,6):
  figure(i)
  plot(d1,N1ex, label="Exact", color='k')

def N1(d):
  return 0.2*d[0]**3 - x*d[1]**2 + 6*d[0]

def N2(d):
  return d[1] - d[0]

def F1ext(n):
  return n*0.25

def F2ext(n):
  return 0

def StiffnessMatrix(d):
  return array([ [0.6*d[0]**2 + 6, -2*x*d[1]], [-1, 1] ])

def dGds(d, Dd, s):
  return -Dd[0]**2*(0.6*(d[0]+s*Dd[0])**2 - 2*x*(d[1]+s*Dd[1]) + 6)

def Fint(d):
  return array([N1(d), N2(d)])

def NR():
  d = zeros((2,N))
  N1_NR = zeros(N)
  iterations_convergence = [0]

  unconverged = False
  for n in range(1, N):
    if unconverged:
      break
    d[:, n] = d[:,n-1]
    di = d[:, n]
    Fext = array([F1ext(n), F2ext(n)])
    R0 = Fext - Fint(di)
    R = R0
    for i in range(0, 15):
      K = StiffnessMatrix(di)
```

```
      Dd = linalg.solve(K, R)

      di += Dd
      R = Fext - Fint(di)
      if (norm(R) < epsilon*norm(R0)):
        iterations_convergence.append(i+1)
        break
      if (i == 14):
        unconverged = True

    d[:, n] = di

    N1_NR[n] = N1(d[:,n])

  figure(1)
  max_load_step = len(iterations_convergence)
  plot(d[0,0:max_load_step], N1_NR[0:max_load_step], '-s', linewidth=2,
      markersize=5, label="Newton-Rhapson")
  figure(6)
  plot(range(0,max_load_step),iterations_convergence, label="Newton-↩
      Rhapson")

def MNR():
  d = zeros((2,N))
  N1_NR = zeros(N)
  iterations_convergence = [0]

  unconverged = False
  for n in range(1, N):
    if unconverged:
      break
    d[:,n] = d[:,n-1]
    di = d[:, n]
    Fext = array([F1ext(n), F2ext(n)])
    R0 = Fext - Fint(di)
    R = R0
    K = StiffnessMatrix(di)
    for i in range(0, 15):
      Dd = linalg.solve(K, R)

      di += Dd
      R = Fext - Fint(di)
      if (norm(R) < epsilon*norm(R0)):
        iterations_convergence.append(i+1)
        break
      if (i == 14):
        unconverged = True

    if (n < N):
      d[:, n] = di

    N1_NR[n] = N1(d[:,n])

  figure(2)
  max_load_step = len(iterations_convergence)
  plot(d[0,0:max_load_step], N1_NR[0:max_load_step], '-s', linewidth=2,
      markersize=5, label="Mod N-R")
```

```
  figure(6)
  plot(range(0,max_load_step),iterations_convergence, label="Mod N-R")

def MNR_LS():
  d = zeros((2,N))
  N1_NR = zeros(N)
  iterations_convergence = [0]

  unconverged = False
  for n in range(1, N):
    if unconverged:
      break
    d[:,n] = d[:,n-1]
    di = d[:, n]
    Fext = array([F1ext(n), F2ext(n)])
    R0 = Fext - Fint(di)
    R = R0
    K = StiffnessMatrix(di)
    for i in range(0, 15):
      Dd = linalg.solve(K, R)
      def G(s):
        return dot(Dd, Fext - Fint(di + s*Dd))
      def dGds(s):
        return -dot(Dd, dot(StiffnessMatrix(di + s*Dd), Dd))

      s0 = 0; s1=1
      while (G(s0)*G(s1) > 0):
        s0 = s1; s1 = 2*s1

      s = optimize.newton(G, s1, fprime=dGds, tol=0.5*abs(G(0)), maxiter=5)

      di += s*Dd
      R = Fext - Fint(di)
      if (norm(R) < epsilon*norm(R0)):
        iterations_convergence.append(i+1)
        break
      if (i == 14):
        unconverged = True

    if (n < N):
      d[:, n] = di

    N1_NR[n] = N1(d[:,n])

  figure(3)
  max_load_step = len(iterations_convergence)
  plot(d[0,0:max_load_step], N1_NR[0:max_load_step], '-s', linewidth=2,
      markersize=5, label="Mod N-R Line Search")
  figure(6)
  plot(range(0,max_load_step),iterations_convergence,
  label="Mod N-R Line Search")

def MNR_BFGS():
  d = zeros((2,N))
  N1_NR = zeros(N)
  iterations_convergence = [0]
```

```
  unconverged = False
  for n in range(1, N):
    if unconverged:
      break
    d[:,n] = d[:,n-1]
    di = d[:, n]
    Fext = array([F1ext(n), F2ext(n)])
    R0 = Fext - Fint(di)
    R = R0
    Rp = R
    K = StiffnessMatrix(di)
    v = zeros((2,15))
    w = zeros((2,15))
    for i in range(0, 15):

      Rbar = R
      for k in range(0, i):
        Rbar = Rbar + dot(v[:,i-k-1], Rbar)*w[:,i-k-1]

      Ddbar = linalg.solve(K, Rbar)

      for k in range(0,i):
        Ddbar = Ddbar + dot(w[:,k], Ddbar)*v[:,k]

      Dd = Ddbar

      di += Dd
      R = Fext - Fint(di)
      DR = R - Rp

      alpha = sqrt(-dot(DR, Dd)/dot(Rp,Dd))
      v[:,i] = Dd/dot(Dd, DR)
      w[:,i] = -DR + alpha*Rp
      Rp = R
      if (norm(R) < epsilon*norm(R0)):
        iterations_convergence.append(i+1)
        break
      if (i == 14):
        unconverged = True

    if (n < N):
      d[:, n] = di

    N1_NR[n] = N1(d[:,n])

  figure(4)
  max_load_step = len(iterations_convergence)
  plot(d[0,0:max_load_step], N1_NR[0:max_load_step], '-s', linewidth=2,
      markersize=5, label="Mod N-R BFGS")
  figure(6)
  plot(range(0,max_load_step),iterations_convergence, label="Mod N-R BFGS"↩
      )

def MNR_BFGS_LS():
  d = zeros((2,N))
  N1_NR = zeros(N)
```

```
iterations_convergence = [0]

unconverged = False
for n in range(1, N):
  if unconverged:
    break
  d[:,n] = d[:,n-1]
  di = d[:, n]
  Fext = array([F1ext(n), F2ext(n)])
  R0 = Fext - Fint(di)
  R = R0
  Rp = R
  K = StiffnessMatrix(di)
  v = zeros((2,15))
  w = zeros((2,15))
  for i in range(0, 15):

    Rbar = R
    for k in range(0, i):
      Rbar = Rbar + dot(v[:,i-k-1], Rbar)*w[:,i-k-1]

    Ddbar = linalg.solve(K, Rbar)

    for k in range(0,i):
      Ddbar = Ddbar + dot(w[:,k], Ddbar)*v[:,k]

    Dd = Ddbar

    def G(s):
      return dot(Dd, Fext - Fint(di + s*Dd))
    def dGds(s):
      return -Dd[0]**2*(0.6*(di[0]+s*Dd[0])**2 - 2*x*(di[0]+s*Dd[0]) + ↩
          6)

    s0 = 0; s1=1
    while (G(s0)*G(s1) > 0):
      s0 = s1; s1 = 2*s1

    s = optimize.newton(G, s1, fprime=dGds)

    di += s*Dd
    R = Fext - Fint(di)
    DR = R - Rp

    alpha = sqrt(-s*(G(s) - G(0))/G(0))
    v[:,i] = Dd/(G(s) - G(0))
    w[:,i] = -DR + alpha*Rp
    Rp = R
    if (norm(R) < epsilon*norm(R0)):
      iterations_convergence.append(i+1)
      break
    if (i == 14):
      unconverged = True

  if (n < N):
    d[:, n] = di
```

```
    N1_NR[n] = N1(d[:,n])

  figure(5)
  max_load_step = len(iterations_convergence)
  plot(d[0,0:max_load_step], N1_NR[0:max_load_step], '-s', linewidth=2,
      markersize=5, label="Mod N-R BFGS LS")
  figure(6)
  plot(range(0,max_load_step),iterations_convergence, label="Mod N-R BFGS ↩
      LS")


NR()
MNR()
MNR_LS()
MNR_BFGS()
MNR_BFGS_LS()
for i in range(1,6):
  figure(i)
  legend(loc='best')
  xlabel(r'$d_1$')
  ylabel(r'$N_1(d_1, d_2)$')
  axis([0,d1[-1]+1,0,11])

figure(6)
legend(loc='best')
xlabel('Load Step Number')
ylabel('Iterations to Convergence')

show()
```