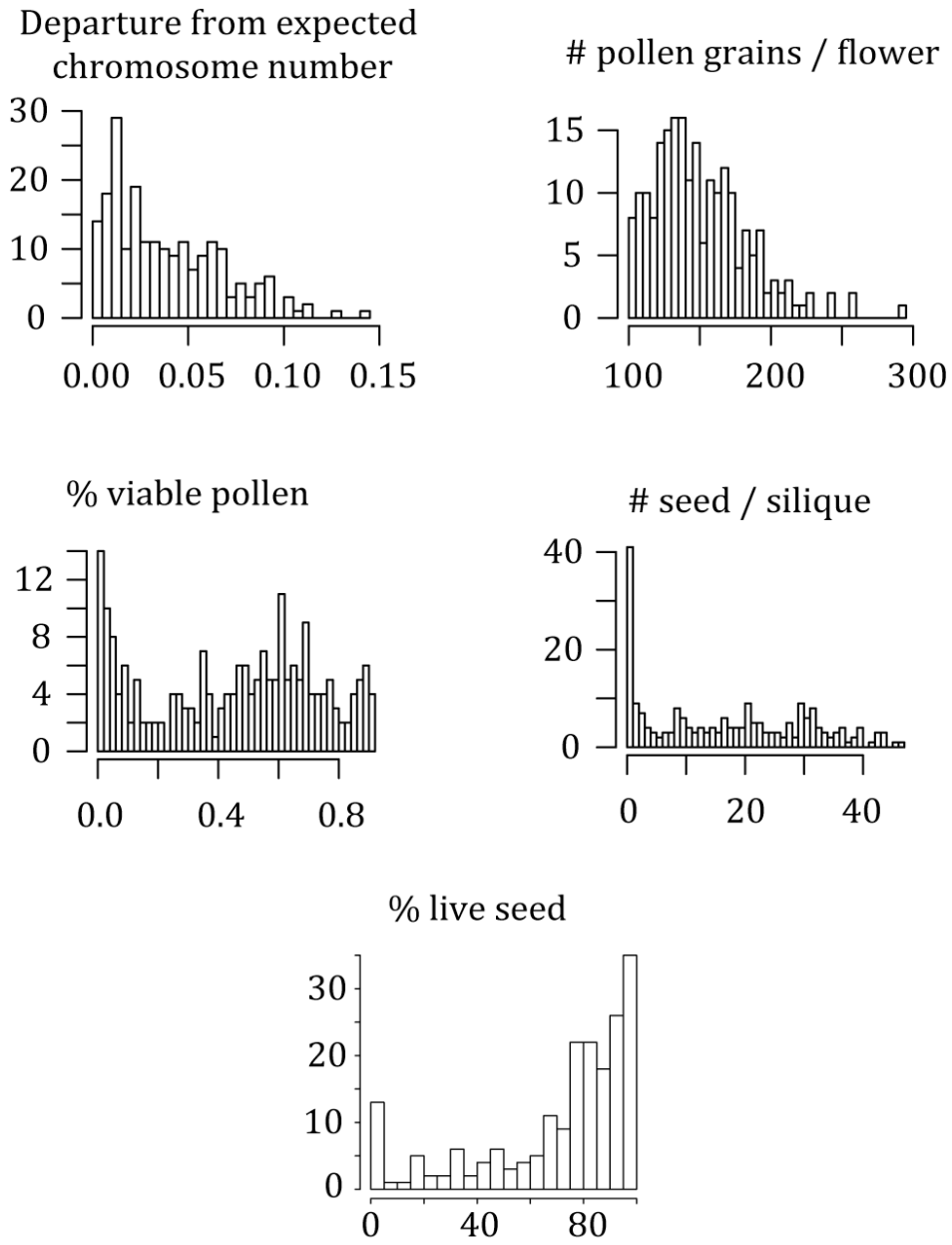
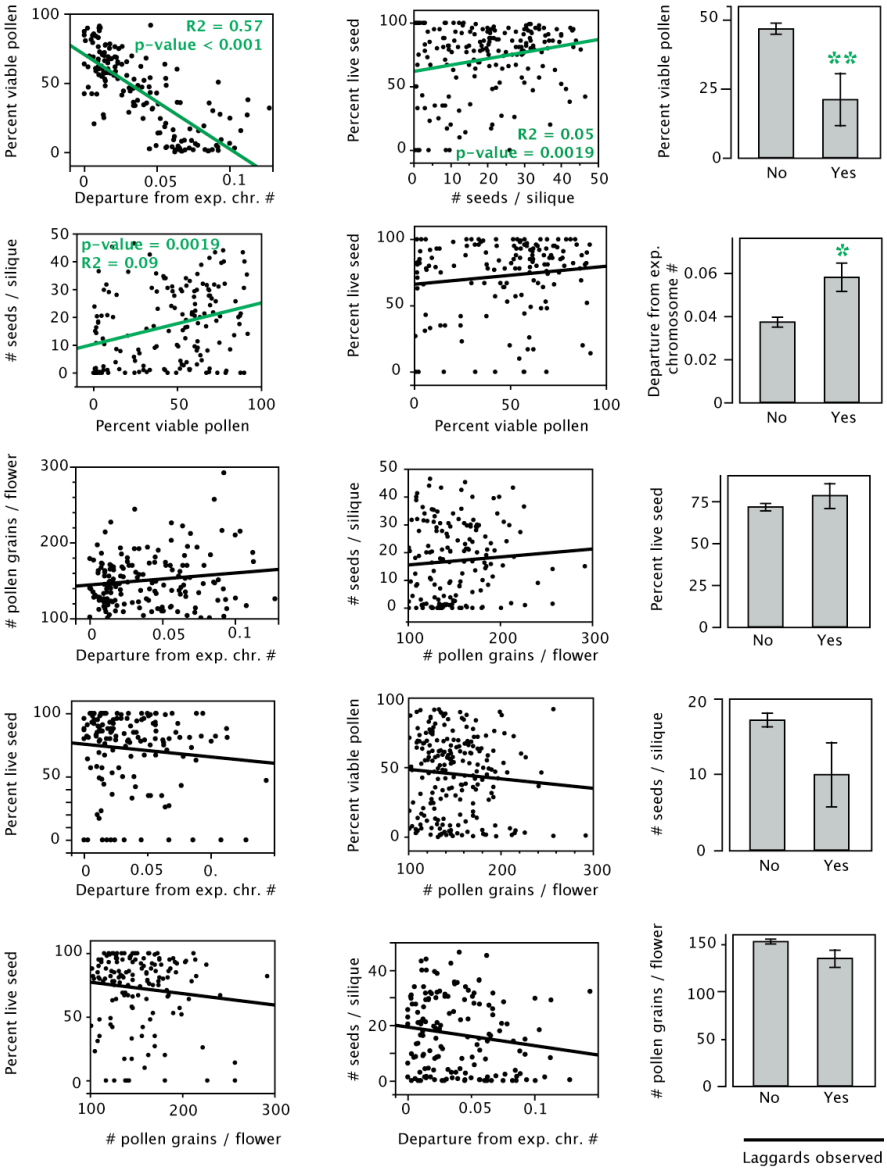
Supplemental Figure 1 - Distribution of phenotype trait values in the Syn x Nat F2 population.



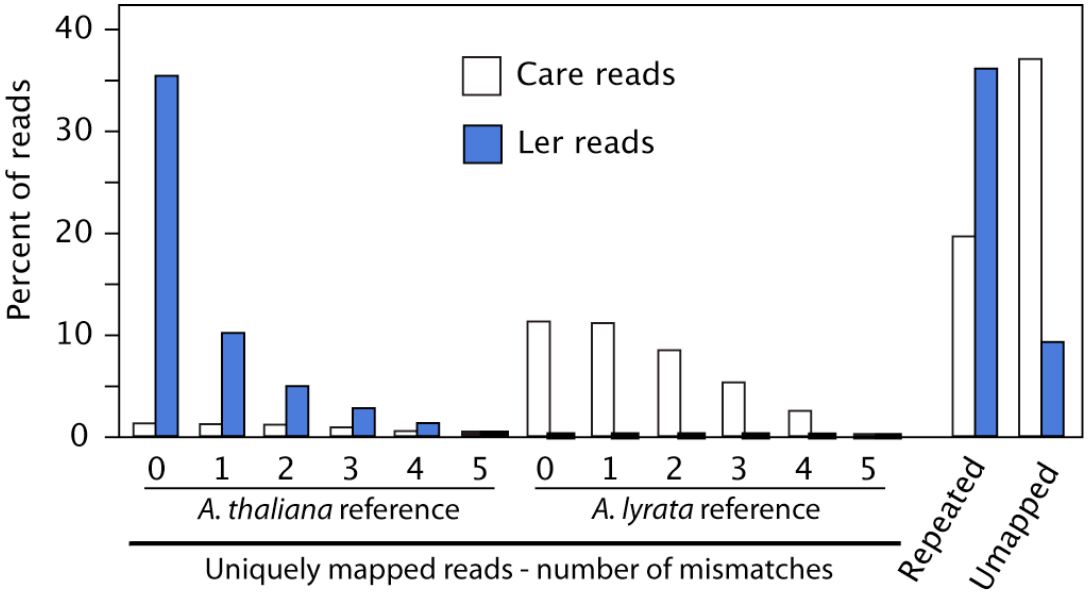
Supplemental Figure 2 – Correlation between the different fertility-related traits in the Syn x Nat F2 population.

For continuous traits, regression p-values and R^2 values for significant regressions (p-values < 0.01) are indicated. For binary traits, the mean value for the second trait is plotted (bars), standard errors are indicated and significant means are indicated ("*" and "**" if the t-test p-value < 0.05 and 0.01 respectively).



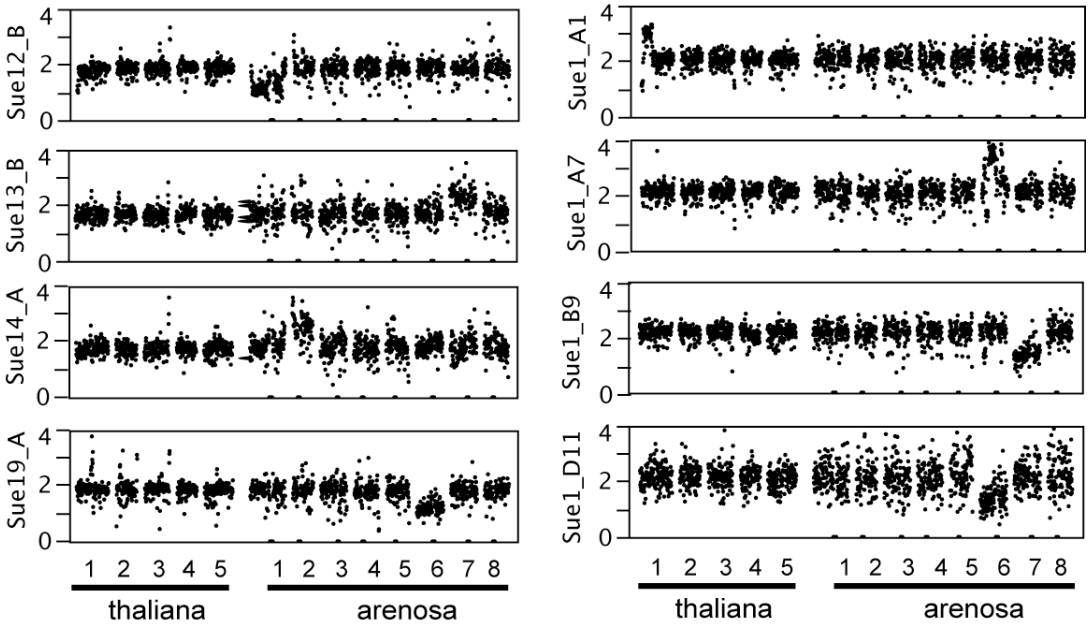
Supplemental Figure 3 – Mapping of parental reads to the hybrid reference genome.

Percentage of sequencing reads preferentially mapping to a single position on the *A. thaliana* or the *A. lyrata* reference, mapping to more than one position ("Repeated") or not mapping ("Unmapped"). For uniquely mapped reads, percentages are further divided according the number of mismatches between the sequencing read and the reference sequence (between zero and five).



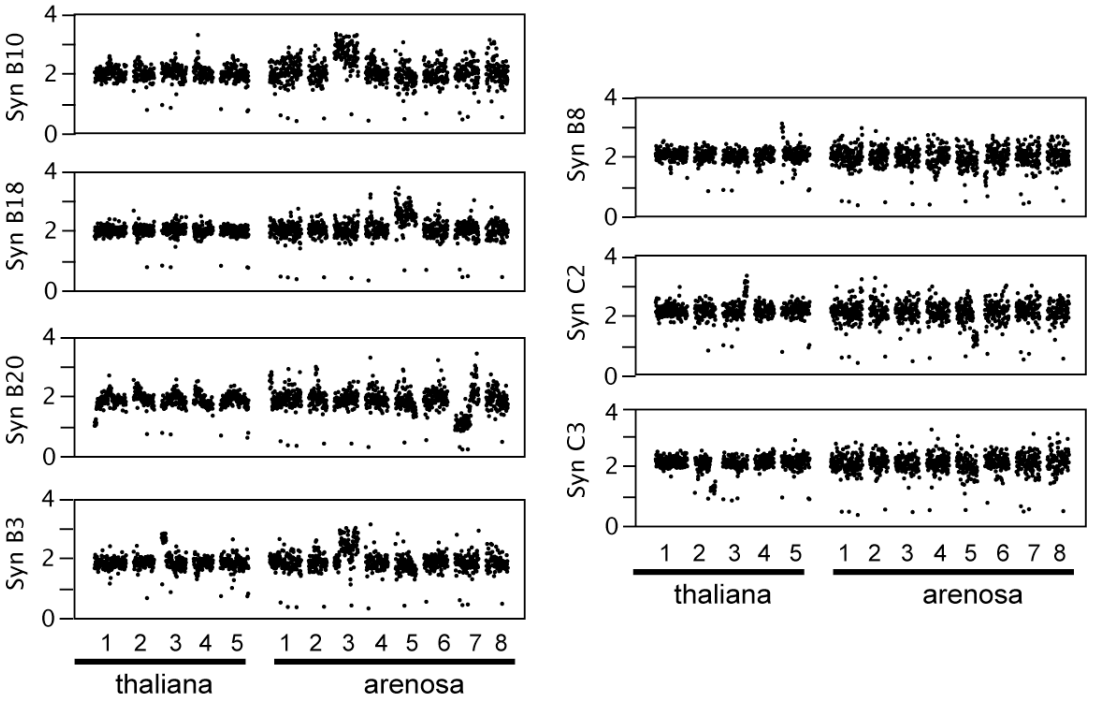
Supplemental Figure 4 – Karyotype of the individuals exhibiting dosage variation in the selfed progeny of *A. suecica* accessions.

A dosage plot was obtained for each individual from whole genome sequencing data. Sequencing reads were aligned to the reference genome and sorted into consecutive bins along the 13 chromosomes of the allopolyploid genome. For ease of visualization, relative read coverage is normalized such that fragments present in two copies fluctuate around 2.0. Changes up or down in relative coverage of consecutive bins indicates variation in dosage and correspond to the addition or deletion of a particular chromosome or chromosomal fragment, respectively. Relative coverage around the centromeric repeats consistently appears noisy most likely because of mis-mapping and poor quality of reference sequence.

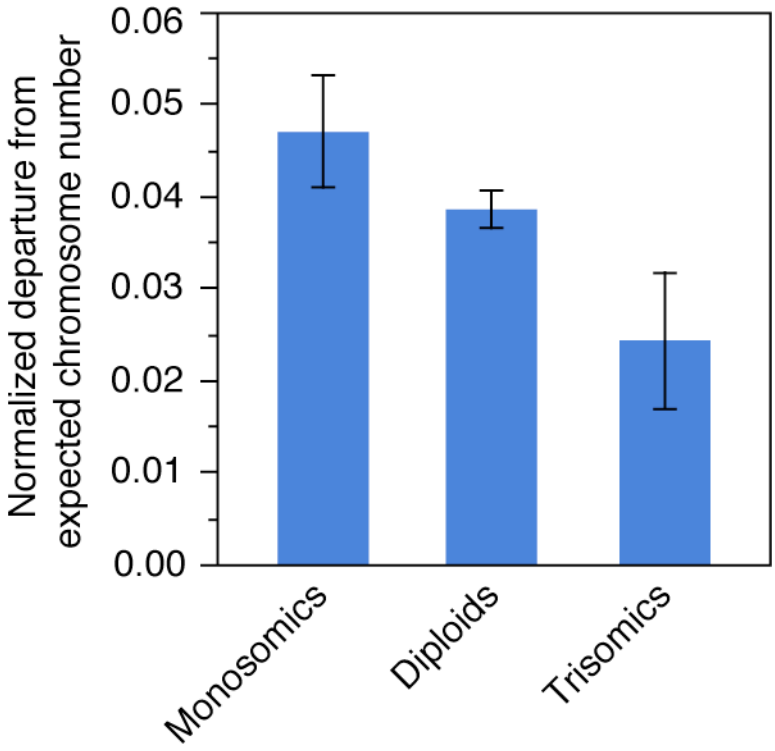


Supplemental Figure 5 – Karyotype of the individuals exhibiting dosage variation in the selfed progeny of synthetic allopolyploids (Syn) individuals

A dosage plot was obtained for each individual from whole genome sequencing data. Sequencing reads were aligned to the reference genome and sorted into consecutive bins along the 13 chromosomes of the allopolyploid genome. For ease of visualization, relative read coverage is normalized such that fragments present in two copies fluctuate around 2.0. Changes up or down in relative coverage of consecutive bins indicates variation in dosage and correspond to the addition or deletion of a particular chromosome or chromosomal fragment, respectively. Relative coverage around the centromeric repeats consistently appears noisy most likely because of mis-mapping and poor quality of reference sequence.

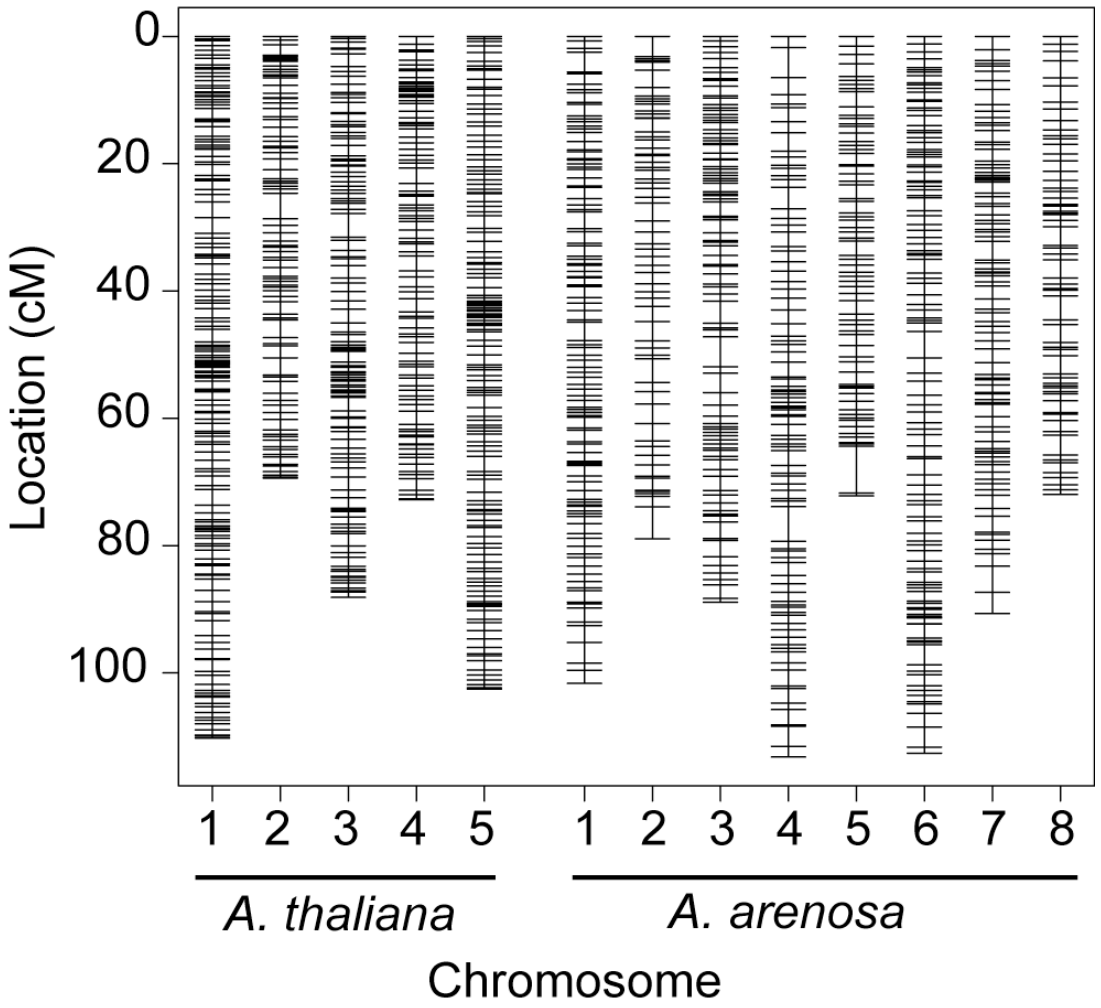


Supplemental Figure 6 – FISH and karyotyping data are consistent with each other. Individuals were classified into three categories according to the results of our karyotyping using next-generation sequencing: samples exhibiting one full additional chromosome (trisomics, N = 10), samples missing one full chromosome (monosomics, N = 10) and true diploids (N = 208). The other samples were excluded from this analysis. For each category, the mean value of our measure of meiotic regularity (mean departure from expected chromosome number, see Methods) was calculated. As expected, the monosomics were different from each other and positioned on either side of the true diploids, which exhibited intermediate values. The means were not significantly different for those comparisons. Standard deviations are shown.



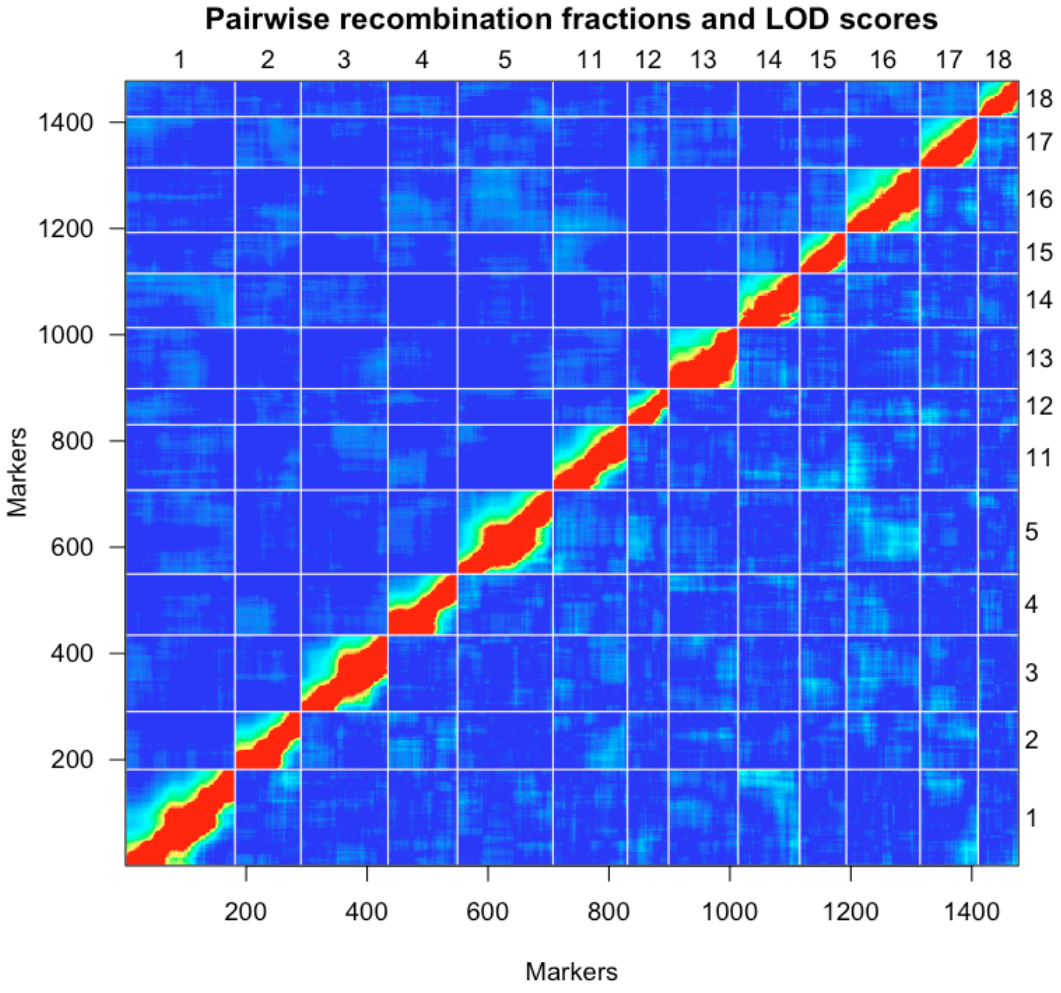
Supplemental Figure 7 - Genetic map of the Syn x Nat F2 population.

Distances are represented in centimorgans (cM).



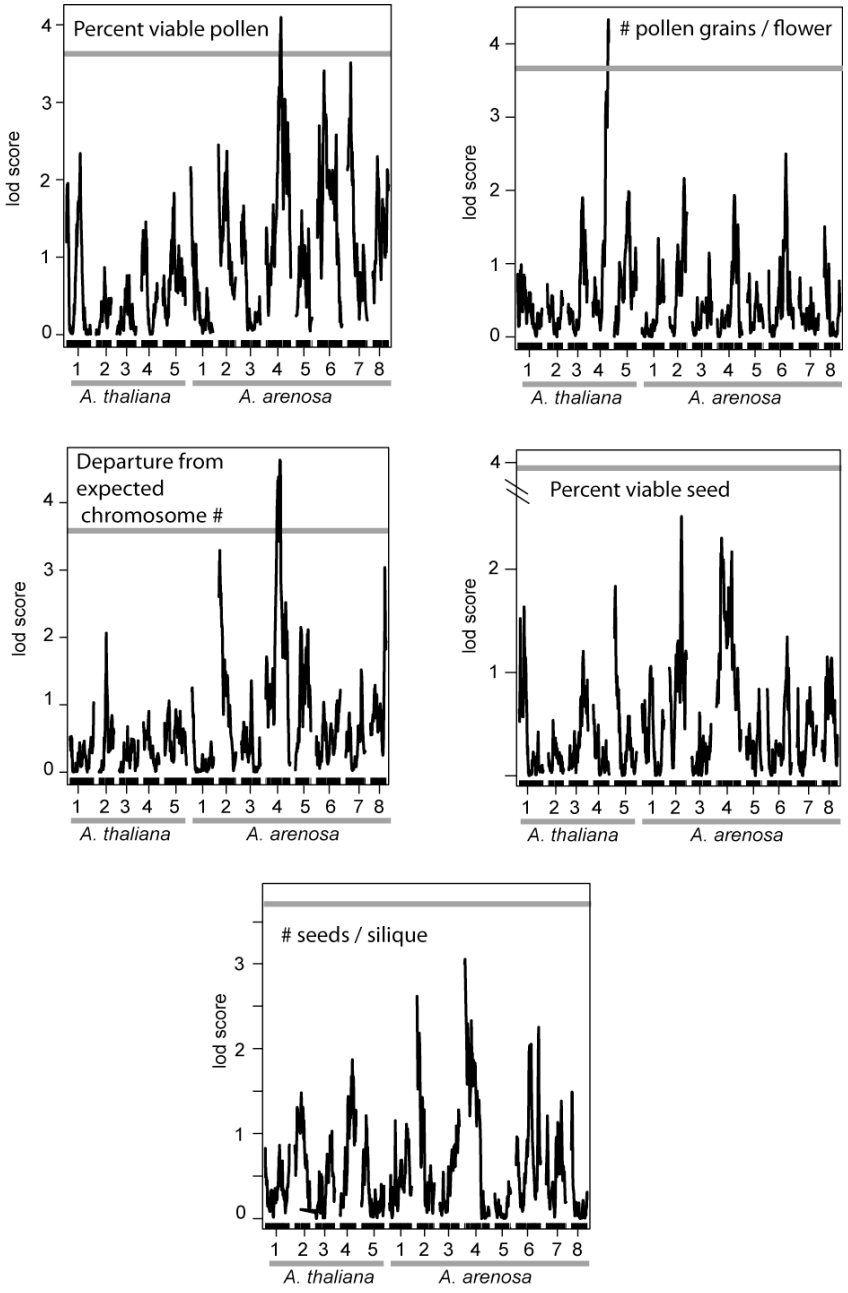
Supplemental Figure 8 – Pairwise recombination frequencies between marker bins.

Continuous color scale, in which red indicates pairs of marker bin that are linked and blue indicated that they are unlinked.



Supplemental Figure 9 – Result of QTL mapping for all traits analyzed.

Results of QTL detection using standard interval mapping in R/qtl. LOD scores are plotted along the five *A. thaliana* and the eight *A. arenosa* chromosomes. Significance thresholds were determined based on 1,000 permutations and correspond to p-values < 0.05.



Supplemental Table 1 – List of sequencing barcodes used for library demultiplexing.

Each column represents one lane of Illumina sequencing.

#	Barcode seq.	SueSyn1	SueSyn2	P 1	P2	F2_1	F2_2	F2_3	F2_4	F2_5	P3	P4
1	AAACA	Sue-13A	Sue1_A_23	Care-1			284	599	14	225		
2	AAATC	Sue-13B	Sue1_B_9		Sue-1		504	246	513	261		
3	AACAT	Sue-13C	Sue-12A	Ler-1			574	285	642	362		
4	AAGAG	Sue-13D	Sue1_D_12			212	506	201	348	103		
5	AAGCC	Sue-13E	Syn-A1			586	287	376	176	188		
6	AAGTT	Sue-13F	Syn-F20			505	524	158	527	146	Ler	
7	AATGC	Sue-13G	Sue1_B_6			620	155	352	567	35		
8	ACACT	Sue-14A	Sue1_B_7			294	646	589	604	76		Sue-1
9	ACCCC	Sue-14B	Sue1_B_8			516	309	375	95	152		
10	ACCGG	Sue-14C	Sue1_D_11			109	18	200	189	300		
11	ACGCA	Sue-14D	Syn-C8			314	600	78	248	159		
12	ACGTG	Sue-14E	Syn-C9			221	32	274	370	242		
13	ACTAA	Sue-14F	Syn-F12			306	174	163	51	47		
14	ACTCG	Sue-14G	Syn-F14			552	577	85	71	10		
15	ACTGT	Sue-14H	Syn-F15			134	183	643	319	283		
16	AGCGA	Sue-14I	Sue1_A_1			514	508	535	167	111		
17	AGCTT	Sue-16A	Sue1_A_14			539	186	349	213	62		
18	AGGAC	Sue-16B	Sue1_A_15			340	526	43	553	116		
19	AGTCA	Sue-16C	Sue1_A_18			545	56	297	17	3		
20	AGTGG	Sue-16D	Sue1_A_20			624	36	2	122	25		
21	ATACG	Sue-16E	Sue1_A_3			378	561	517	550	101		
22	ATAGC	Sue-19A	Sue1_A_5			289	501	384	120	165	Care1	
23	ATCAC	Sue-19C	Sue1_B_3			573	295	204	45	110		
24	ATGTA	Sue-1_18	Sue1_B_5			559	548	533	223	363		
25	ATTCC	Sue-19B	Sue1_D_21			582	264	273	19	68		
26	ATTGA	Sue1_A_2	Syn-B18			557	555	277	241	22		
27	CAAAA	Sue1_A_8	Syn-B19			80	244	154	328	53		
28	CAATT	Sue1_A_9	Syn-B20			227	562	615	83	325		
29	CACCC	Sue1_A_10	Sue-12B			610	371	222	9	153		

30	CATCT	Sue1_A_11	Syn-C11			605	293	196	46	40		
31	CATTC	Sue1_A_16	Syn-C12			618	299	594	347	209		
32	CCCCA	Sue1_A_19	Syn-C14			565	255	329	253	301	Ler	
33	CCCTT	Sue1_D_13	Syn-F1			193	296	141	44	236		
34	CCGAG	Sue1_D_15	Syn-F10			317	239	549	645	65		
35	CGAAT	Sue1_D_16	Syn-F11			580	547	119	366	138		
36	CGAGG	Sue1_D_17	Syn-F13			609	616	139	156	67		
37	CGATC	Sue1_D_20	Syn-F17			131	621	522	333	206		
38	CGCAC	Sue1_D_22	Syn-F19			518	57	585	102	50		Sue-1
39	CGCGT	Sue1_D_23	Syn-F2			543	54	127	566	534		
40	CGGTT	Sue1_D_24	Sue-12C			173	560	177	316	96		
41	CGTTG		Syn-F7			593	596	121	112	237		
42	CTAAG		Sue1_A_6			509	602	313	182	64		
43	CTCGA		Syn-F8			607	81	94	648	28	Care1	
44	CTCTC		Sue1_A_17			540	364	354	128	29		
45	CTGCC		Sue1_A_21			268	88	523	619	228		
46	CTGGT		Sue1_A_22			512	178	235	198	147		
47	CTTGG		Sue1_A_24			544	216	611	217	172		
48	GAAGC		Sue1_A_4			263	5	73	575	42		
49	GACTT		Sue1_A_7			247	207	568	26	166		
50	GAGCA		Sue1_B_1			308	69	48	262	49		
51	GATAC		Sue1_B_2			249	199	368	79	286		
52	GATGA		Sue1_D_10			689	292	144	7	130		
53	GCAAA		Sue1_D_19			520	603	266	33	327		
54	GCCAT		Syn-B13			608	132	187	136	224		
55	GCCTA		Syn-B14			282	257	350	197	1		
56	GCGTT		Syn-B15			260	373	233	302	30		
57	GCTCC		Syn-B16			31	576	175	11	320		
58	GCTGG		Syn-B17			117	4	612	341	195		
59	GGAAC		Syn-B25			137	374	626	38	191	Ler	
60	GGATG		Syn-B3			541	529	168	635	298	Care1	
61	GGCAG		Syn-B4			510	86	135	20	185		
62	GGGAA		Syn-C13			647	98	105	97	84		
63	GGTGT		Syn-F18			515	270	305	140	160		
64	GTACC		Syn-F5			591	238	27	55	107		
65	GTAGT		Sue1_A_12			149	357	90	6	269		
66	GTATA		Sue1_A_1			351	372	226	536	106		

			3									
67	GTCAA		Syn-B5			164	240	617	59	37		
68	GTCCT		Syn-B6			551	583	126	8	331		
69	GTCTG		Syn-B7			532	61	571	211			
70	GTGAG		Syn-B21			143	587	546	229			Sue-1
71	GTGGA		Syn-B22			503	219	614	365			
72	GTTAT		Syn-C6			332	627	108	124			
73	GTTCA		Sue-2A			355	636	63	265			
74	GTTTC		Syn-B10			323	184	321	336			
75	TAAGA		Syn-B2			337	572	148	315			
76	TACTG		Syn-B8			335	254	346	100			
77	TAGCT		Syn-B11			638	570	525	91			
78	TATAG		Syn-B12			303	334	89	290			
79	TATCC		Syn-B23			613	588	77	563		Ler	
80	TCATT		Syn-B24			330	623	310	622			
81	TCCAG		Syn-C3			641	356	169	114			
82	TCCCT		Syn-F6			353	234	52	15			
83	TCGAC		Syn-C2			170	259	23	322			
84	TCGCG		Syn-C7			318	58	343	115			
85	TCGGA		Syn-B1			359	280	564	66			
86	TGAAG		Sue-5A			530	218	342	104			
87	TGACC		Sue-7A			584	367	150	125			
88	TGATA		Syn-F16			271	24	203	252			
89	TGGAT		Sue-2B			640	377	180	171			
90	TGGTC		Sue-2C			93	192	214	272			
91	TGTCG		Sue-2D			181	231	502	162			
92	TGTGC		Syn-B26			157	578	215	145			
93	TGTTT		Sue-2E			194	190	39	511			
94	TTACT		Sue-3A			542	528	99	179		Care1	
95	TTAGG		Sue-4A			500	208	554	230			Sue-1
96	TTCCC		Sue-4B			556	590	644	123			

P: parents. F2: Syn x Nat F2 individuals, SueSyn: Progeny of *A. suecica* (Sue) and synthetic allopolyploid (Syn) individuals.

Supplemental Table 2 – Size of the genetic map

Genome	Chromosome	Length (cM)	# marker bins
A. thaliana	1	110.2	181
	2	69.4	109
	3	88.1	144
	4	72.8	115
	5	102.5	158
A. arenosa	1	101.6	123
	2	78.9	68
	3	88.9	115
	4	113.2	102
	5	72.2	77
	6	112.6	122
	7	90.7	96
	8	72	67
Total	*A. thaliana*	443	707
	A. arenosa	730.1	770

Supplemental Methods 1 – SynxNat1_Screen_mpup.py

Isabelle Henry, last modified June 2012
UC Davis Genome Center
This script takes an mpileup file as input (samtools) that contains allelic calls for Ler (A.thaliana), Care-1 (A. arenosa 4x),
Sue (A. suecica) and the pooled F2 individuals. The mpileup comes from aligning each of these samples to the hybrid reference
genome (tair 9, five chromsoomes + A. lyrata scaffolds 1 to 8). It then select specific positions based on the following criteria:
- covered at least 10 times in the F2 pooled data
- covered at least once in Sue
- If position on A. thaliana reference, it should not be covered in Care but be covered in the Ler
- If position on A. lyrata referemce, it should be covered in Care but not in Ler
- At least one basecall should be different from the reference in the F2 allele calls.

This is just a quick way to reduce the size of the file prior to more elaborate filtering of potential marker positions.

```
import sys

from collections import defaultdict

mpup = open(sys.argv[1])
o = open(sys.argv[2],'w')

counts = defaultdict(int)

for line in mpup:
    if line[0] == 'C':
        o.write(line)
    else:
        l = line.split()
        chrom = l[0]
        counts[chrom] += 1
        if l[12] != '0' and int(l[9]) > 9:
            if l[0][1] == 'l' and l[3] == '0' and l[6] != '0':
                if 'A' in l[10] or 'C' in l[10] or 'G' in l[10] or 'T' in l[10] or
'a' in l[10] or 'c' in l[10] or 'g' in l[10] or 't' in l[10] or '+' in l[10] or '*'
in l[10]:
                    o.write(line)
                    counts[str(chrom)+'kept'] += 1
            elif l[0][1] == 't' and l[3] != '0' and l[6] == '0':
                if 'A' in l[10] or 'C' in l[10] or 'G' in l[10] or 'T' in l[10] or
'a' in l[10] or 'c' in l[10] or 'g' in l[10] or 't' in l[10] or '+' in l[10] or '*'
in l[10]:
                    o.write(line)
                    counts[str(chrom)+'kept'] += 1
numbers = counts.keys()
numbers.sort

for x in numbers:
    print str(x) + '\t' + str(counts[x])

o.close()
mpup.close()
```

Supplemental Methods 2 – SynxNat2_ Screen_parsed_mpup.py

```
# Isabelle Henry, last modified June 2012
# UC Davis Genome Center

# This script takes a parsed mpileup file as input (samtools) that contains allelic
calls for Ler (A.thaliana), Care-1 (A. arenosa 4x),
# Sue (A. suecica) and the pooled F2 individuals. The mpileup comes from aligning
each of these samples to the hybrid reference
# genome (tair 9, five chromsoomes + A. lyrata scaffolds 1 to 8). It then select
specific positions based on the following criteria:
# 10 < F2 coverage < 300
# If position on A. thaliana reference:
#    - No coverage in Care
#    - Covered at least five times in Ler and Sue.
#    - The most common allele from Ler and Sue represent > 90% of their base calls
(not heterozygous) and are different from each other
#    - The same two alleles are represented in the F2, each represent at least 20% of
the base call for the pooled F2 population and represent at least 90% of the total
together
# Similar criteria apply for the A. arenosa positions.

import sys

from collections import defaultdict

pmpup = open(sys.argv[1])
o = open(sys.argv[2],'w')

counts = defaultdict(int)

for line in pmpup:
    if line[0] == 'C':
        o.write(line)
    else:
        l = line.split()
        chrom = l[0]
        counts[chrom] += 1

##        0 = chr,
##        3 = SNP1Ler,
##        6 = CovLer,
##        7 = SNP1Care
##        10 = covCare,
##        12 = SNP2-F2,
##        14 = covF2,
##        15 = SNP1-Sue
##        18: cov Sue,

        if l[18] != '.' and l[14] != '.' and l[12] != '.':
            if l[0][1] == 't' and l[6] != '.' and l[10] == '.':
                syn1 = l[3]
                syncov = int(l[6])
                syn = 'Ler'
            elif l[0][1] == 'l' and l[6] == '.' and l[10] != '.':
                syn1 = l[7]
                syncov = int(l[10])
```

```
                syn = 'Care'
            else:
                continue
            Sue1 = l[15]
            Suecov = int(l[18])
            F2cov = int(l[14])
            ## if thaliana position, Care has not be blank, Ler and Sue must be
homozygous (>= 90%)
            if syn1.split('_')[0] != Sue1.split('_')[0]:
                if float(syn1.split('_')[1]) >= 90 and float(Sue1.split('_')[1])>=
90:
                   if Suecov > 5 and syncov > 5 and 300 > F2cov > 10 and l[12] !=
'.':
                       F2A1 = l[11].split('_')[0]
                       F2A2 = l[12].split('_')[0]
                       F2p1 = l[11].split('_')[1]
                       F2p2 = l[12].split('_')[1]
                       if float(F2p1) > 20 and float(F2p2) > 20 and float(F2p1) +
float(F2p2) > 90:
                           if syn1.split('_')[0] in [F2A1, F2A2] and
Sue1.split('_')[0] in [F2A1, F2A2]:
                               o.write(line)
                               counts[chrom+'yes'] += 1

numbers = counts.keys()
numbers.sort

for x in numbers:
    print str(x) + '\t' + str(counts[x])

o.close()
pmpup.close()
```

Supplemental Methods 3 – SynxNat3_CallAlleles.py

```
# Isabelle Henry, last modified June 2012
# UC Davis Genome Center

# This script takes a parsed mpileup file as input (samtools) that contains allelic
calls for Ler (A.thaliana), Care-1 (A. arenosa 4x),
# Sue (A. suecica) and the pooled F2 individuals. The mpileup comes from aligning
each of these samples to the hybrid reference
# genome (tair 9, five chromsoomes + A. lyrata scaffolds 1 to 8).

# The mpileup file has already been screened for robust marker positions in the
previous steps.

# This script compared the alleles in the two parental genotypes and assign an allele
for the F2 individuals
# The criteria are as follows:
# - if only one allele type is found, and that allele corresponds to one of the two
parental allele, that allele is called.
# - if there are two allele types and the least frequent allele represents < 10% of
the calls, it is treated as if only the most common type was present
# - if there are two allele types and the least frequent allele represents > 10% of
the allele calls, and the two alleles
# correspond to the two parents, the allele is called Het.
# - in all other cases, the genotype is called "na"

# USAGE: SynxNat3_CallAlleles.py parsed_mpileup_marker.txt output.txt


import sys

pos = open(sys.argv[1])
o = open(sys.argv[2],'w')

indices = []
totalcount = 0
odd1 = 0
odd2 = 0

for line in pos:
    l = line.split()
    if line[0] == 'C':
        count = 0
        header = 'Chrom\tPos\tRef\tSyn\tSue'
        for m in l:
            if m.split('-')[0] == 'Cov':
                indices.append(count)
                lib = m.split('-')[1]
                header += '\tSnptype-'+lib+'\tSNP1-'+lib+'\tSNP2-'+lib+'\tSNPCov-
'+lib+'\tAll-'+lib
            count += 1
        o.write(header+'\n')
        print indices
    else:
        if l[0][1] == 't':
            syn = l[3].split('_')[0]
            genome = 't'
        elif l[0][1] == 'l':
```

```
            syn = l[7].split('_')[0]
            genome = 'l'
        Sue = l[15].split('_')[0]
        text = l[0]+'\t'+l[1]+'\t'+l[2]+'\t'+syn+'\t'+Sue
        for i in indices:
            Cov = l[i]
            if Cov == '.':
                text += 5 * '\t.'
            else:
                totalcount += 1
                snp1 = l[i-3]
                snp2 = l[i-2]
                snp3 = l[i-1]
                if snp3 != '.':
                    Snptype = 3
                elif snp2 != '.':
                    Snptype = 2
                else:
                    Snptype = 1
                Snp1 = snp1.split('_')[0]
                if snp2 == '.':
                    Snp2 = '.'
                elif float(snp2.split('_')[1]) <= 10:
                    Snp2 = '.'
                else:
                    Snp2 = snp2.split('_')[0]
                if (Snp1 == syn and Snp2 == Sue) or (Snp1 == Sue and Snp2 == syn):
                    All = 'H'
                elif Snp2 != '.':
                    All = '.'
                    odd2 += 1
                elif Snp1 == syn:
                    All = 'Y'
                elif Snp1 == Sue:
                    All = 'S'
                else:
                    All = '.'
                    odd1 += 1
                text +=
'\t'+str(Snptype)+'\t'+str(Snp1)+'\t'+str(Snp2)+'\t'+str(Cov)+'\t'+All
        text += '\n'
        o.write(text)

o.close()
pos.close()

print totalcount
print odd1
print odd2
```

Supplemental Methods 4 – SynxNat4_ Create_MarkerBins.py

```
# Isabelle Henry, last modified June 2012
# UC Davis Genome Center

# This script takes a parsed mpileup file as input (samtools) that contains allelic
calls for Ler (A.thaliana), Care-1 (A. arenosa 4x),
# Sue (A. suecica) and the pooled F2 individuals. The mpileup comes from aligning
each of these samples to the hybrid reference
# genome (tair 9, five chromsoomes + A. lyrata scaffolds 1 to 8).

# The mpileup file has already been screened for robust marker positions in the
previous steps.

# This script created MarkerBins, that encompass x (user define) adjacent markers and
calculated a mean genotype for each of the sample
# at each of the marker bins. Marker bins are non-overlapping and are positioned next
to each other along each of the chromosomes.

# For each marker bin, the start and end position (first and last marker in the bin)
are recorded, as well as the middle position.
# For each sample in the mpileup file, genotype at the marker bin is determined
according to the following criteria:
# - if less than 5 markers are genotyped within a window, the markerbin genotype is
called "na"
# - if at least 5 are present, an average genotype is derived using the following
rule: An S (Sue) allele counts for 1, a Y (synthetic) allele (ler or care)
# counts for 0, a H (het) allele counts for 0.5
# - If the mean is < 0.05, the allele is called Y (synthetic)
# - If the mean is > 0.95, the allele is called S (Sue)
# - If 0.25<mean<0.75, the allele is called H (het).
# - Else, the allele is called "na"

# for each marker a mean % Sue is also calculated.

# a second file is also output that contains that actual mean % Sue for each window
and each sample


# USAGE: SynxNat4_Create_Markerbins.py parsed_mpup.txt output.txt windowsize

import sys

five = open(sys.argv[1])
o = open(sys.argv[2],'w')
o2 = open(sys.argv[2].split('.txt')[0]+'_mean.txt', 'w')
window = int(sys.argv[3])

indices = []
libwin = {}
Pos = []

chrom = 'AtChr1'
N = 0


# processes the header line to determine the columns numbers and which samples
correspond to them
```

```
for line in five:
    l = line.split()
    if line[0] == 'C':
        count = 0
        header = 'Chrom\tPos\tBps'
        for m in l:
            if m.split('-')[0] == 'All':
                lib = m.split('-')[1]
                indices.append((count,lib))
                header += '\t'+lib
            count += 1
        header += '\t#Y\t#S\t#H\t#Low\t#High\t#na\tTotal\t%na\t%Sue\n'
        o.write(header)
        o2.write(header)
        print indices
        numlib = len(indices)
        print numlib
        for i in indices:
            libwin[i[1]] = []

#
    else:
        Chrom = l[0]
        if N == 0:
            chrom = l[0]
        if Chrom == chrom and N < window:
            N += 1
            Pos.append(l[1])
            for i in indices:
                allele = l[i[0]]
                if allele == 'Y':
                    libwin[i[1]].append(0)
                elif allele == 'S':
                    libwin[i[1]].append(1)
                elif allele == 'H':
                    libwin[i[1]].append(0.5)
        if N == window or Chrom != chrom:
            if N == window and len(Pos) != window:
                print 'PROBLEM'
                break
            countna = 0
            countzero = 0
            countone = 0
            counthalf = 0
            countlow = 0
            counthigh = 0
            region = int(Pos[-1])-int(Pos[0])
            MidPos = int(Pos[-1])-region/2
            output = str(chrom)+'\t'+str(MidPos)+'\t'+str(region)
            outputmean = str(chrom)+'\t'+str(MidPos)+'\t'+str(region)
            for j in indices:
                if len(libwin[j[1]]) < 5:
                    output += '\t.'
                    outputmean += '\t.'
                    countna += 1
                else:
                    average = float(sum(libwin[j[1]])) / len(libwin[j[1]])
                    outputmean += '\t'+str(average)
                    if average < 0.05:
                        output += '\t0'
```

```
                        countzero += 1
                    elif average > 0.95:
                        output += '\t1'
                        countone += 1
                    elif 0.25 < average < 0.75:
                        output += '\t0.5'
                        counthalf +=1
                    elif average <= 0.25:
                        output += '\t0.25'
                        countlow += 1
                    elif average >= 0.75:
                        output += '\t0.75'
                        counthigh += 1
                    else:
                        print average
                        break
                libwin[j[1]] = []
            if N > 25:
                total = countzero + countone + counthalf + countlow + counthigh +
countna
                percna = 100.00 * countna / total
                SueRatio = (float(countone) + (float(counthalf) / 2.00)) /
float(countzero + countone + counthalf)
                output +=
'\t'+str(countzero)+'\t'+str(countone)+'\t'+str(counthalf)+'\t'+str(countlow)+'\t'+st
r(counthigh)+'\t'+str(countna)+'\t'+str(total)+'\t'+str(percna)+'\t'+str(SueRatio)+'\
n'
                print chrom
                N = 0
                Pos = []
                o.write(output)
                o2.write(outputmean+'\n')
            else:
                N = 0
                Pos = []
o.close()
five.close()
```

Supplemental Methods 5 – SynxNat5_Imputation.py

```
# Isabelle Henry, last modified June 2012
# UC Davis Genome Center

# This script takes an r-qtl genotype file, in which marker oder has been optimized
and imputes missing genotypes
# as well as removed outlier genotypes.
# The criteria are as follows:
# For the middles of the chromosomes: the script looks for windows of maximum 15
markers, in which the first, last and one more are the same genotype
# it then looks for other genotypes. It there is more than one, it leaves the window
as is. If there is one or none, it fills the window
# with that genotype.

# For the beginning of the chromosomes: if the first three assigned markers agree,
are less than 14 markers apart, and they are less than 14 markers away from the
beginning,
# then the first unassigned markers get filled with the same genotype as the first
three assigned markers

# For the end of the chromosomes: if the window is maximum 10 markers long, the first
marker of the window and the one before that are consistent an
# and there is no other call in the window, the window is filled with that marker


# USAGE: SynxNat5_Imputation.py GenotypeFile output.txt pos
# if genetic is used, the script assumes the presence of a third hearder line
containing the position of the markers


import sys

bins = open(sys.argv[1])
o = open(sys.argv[2],'w')

first = []

# marker name lines

markers = bins.readline()
m2 = markers.split('\n')[0].split(',')
print len(m2)
o.write(markers)

chroms = bins.next()
ch = chroms.split('\n')[0].split(',')
print len(ch)
o.write(chroms)

# makes a list of the first column of each chromosome
prevc = '.'
count = 1
for c in ch[1:]:
    if c != prevc:
        first.append(count)
    prevc = c
    count += 1
```

```
print first

# need to add the first column of the next chromosome to identify the end of the last
one

first.append(count)

print first

# write the genetic distance line
if 'pos' in sys.argv:
    line = bins.next()
    o.write(line)

## window function looks for the second instance of the same markers in a window of
15 genotypes
## it returns the position of the third instance of the same marker as the first in
that window
## a value of 2 means that the first three are identical. Can go on to the next
marker
## > 2 means that there are three identical markers but there is also at least one
other
## 0 means that there is not two more markers identical to the first one in that
window

def window(marker, bigwin):
    count = 0
    second = []
    for g in bigwin:
        if g == marker:
            second.append(count)
            count += 1
        else:
            count += 1
    if len(second) > 2:
        return (second[2])
    else:
        return (0)

## going through each F2 individual

onec = zeroc = halfc = nac = 0

for mine in bins:
    m = mine.split('\n')[0].split(',')
    lib = str(m[0])
    geno = {}
    c = 0
    for chrom in first[:-1]:
        geno[chrom] = m[first[c]:first[c+1]]
        c += 1
    Chromo = geno.keys()
    Chromo.sort()
    for n in Chromo:
        b = 'begin'
        col = 0
        ending = len(geno[n])
        for w in range (0,ending):
            marker = geno[n][w]
```

```
            if marker == 'YY' or marker == 'YS' or marker == 'SS':

## to fill in the blanks at the beginning of the chromosomes
                if b == 'begin':
                    FIRST = 'no'
                    if col < 14:
                        First = col
                        b = 'middle'
                    else:
                        b = 'middle'
                    bigwin = geno[n][col:col+15]

## to make smaller windows at the end of the chromosomes
                elif col > len(geno[n])-15:
                    bigwin = geno[n][col:]
                    b = 'end'
## for the middle
                else:
                    bigwin = geno[n][col:col+15]

                if len(bigwin) > 2:
                    end = window(marker, bigwin)
                    if end > 2:
                        win = geno[n][col:col+end+1]
                        if win[0] != win[-1] or win[0] != marker or win.count(marker)
!= 3:
                            break
                        else:
                            All = win.count('SS')+ win.count('YY')+win.count('YS')
                            others = All - win.count(marker)

                            if others < 2:
                                FIRST = 'yes'
                                for d in range (0,len(win)):
                                    x = win[d]
                                    if x == marker:
                                        continue
                                    elif x == 'SS':
                                        onec += 1
                                        win[d] = marker
                                    elif x == 'YS':
                                        halfc += 1
                                        win[d] = marker
                                    elif x == 'YY':
                                        zeroc += 1
                                        win[d] = marker
                                    elif x == '-':
                                        nac += 1
                                        win[d] = marker
                                    else:
                                        print "invalid value"
                                        print x
                                        break
                                if col == First:
                                    for z in range(0,First):
                                        geno[n][z] = marker
                                geno[n][col:col+end+1] = win

## for filling in the end if there are no more markers identical to the first one
                    elif end == 0 and col > (len(geno[n]) - 10):
```

```
                        win = geno[n][col-1:len(geno[n])]
                        if win.count('SS') + win.count('YS') + win.count('YY') -
win.count(marker) == 0 and win.count(marker) > 1:
                            for x in range (col,len(geno[n])):
                                geno[n][x] = marker

## for filling in the very beignning of the chromosome
                    elif col == First:
                        if end == 2:
                            for z in range(0,First):
                                geno[n][z] = marker
                        elif end>2 and FIRST == 'yes':
                            for z in range(0,First):
                                geno[n][z] = marker

## to remove an outlier stuck in a crossing-over (not otherwise removed)
                    elif end == 0 and bigwin.count('SS') > 0 and bigwin.count('YY') >
0 and bigwin.count('YS') > 0:
                        oneb = bigwin.count('SS')
                        zerob = bigwin.count('YY')
                        halfb = bigwin.count('YS')
                        if marker == 'SS' and zerob < halfb and zerob < 5:
                            markerb = 'YY'
                        elif marker == 'SS' and halfb < zerob and halfb < 5:
                            markerb = 'YS'
                        elif marker == 'YY' and halfb < oneb and halfb < 5:
                            markerb = 'YS'
                        elif marker == 'YY' and halfb > oneb and oneb < 5:
                            markerb = 'SS'
                        elif marker == 'YS' and zerob < oneb and zerob < 5:
                            markerb = 'YY'
                        elif marker == 'YS' and oneb < zerob and oneb < 5:
                            markerb = 'SS'
                        else:
                            markerb = '-'
                        for i in range (0, len(bigwin)):
                            if bigwin[i] == markerb:
                                bigwin[i] = '-'
                        if b == 'middle':
                            geno[n][col:col+15] = bigwin
                        elif b == 'end':
                            geno[n][col:] = bigwin

            col += 1

## printing the new genotypes and calculating the percentages of each genotype
    formatter = ''

    for p in range (1,first[0]):
        formatter += ','+m[p]
##        if m[p] == '' or m[p] == '.':
##            m[p] = 'NA'
##        formatterqtl += ','+m[p]

    Syn = Sue = na = het = 0
    for Chrom in Chromo:
        for j in geno[Chrom]:
            if j == 'YY':
                Syn += 1
            elif j == 'SS':
```

```
                Sue += 1
            elif j == '-':
                na += 1
            elif j == 'YS':
                het += 1
            else:
                print "invalig genotype 2"
                print j
                break
            formatter += ','+str(j)
    total = Syn + Sue + het + na
    data = Syn + Sue + het
    if Syn + Sue + het > 0:
        PerSue = 100 * (Sue + het/2) / (Sue + het + Syn)
        PerSyn = 100 * (Syn) / (Sue + het + Syn)
    else:
        PerSue = '.'
        PerSyn = '.'
    if PerSyn > 1:

o.write(lib+formatter+','+str(Sue)+','+str(het)+','+str(Syn)+','+str(na)+','+str(tota
l)+','+str(data)+','+str(PerSue)+',F2\n')
    else:

o.write(lib+formatter+','+str(Sue)+','+str(het)+','+str(Syn)+','+str(na)+','+str(tota
l)+','+str(data)+','+str(PerSue)+',F1xSue\n')
    print "finished" + str(lib)


o.close()
bins.close()
print str(zeroc)
print str(halfc)
print str(onec)
print str(nac)
```